P.S.I

LANGAGES

TURBO PASCAL PAS A PAS

VERSION 4

JEAN-MICHEL GAUDIN

PROGRAMMATION

- Initiation
- Procédures
- Programmes
- Exemples

TURBO PASCAL PAS A PAS

VERSION 4

Connaissez-vous la collection Langages chez PSI ?

Basic
Le livre du Basic sur PC et compatibles - Jacques Boisgontier
Le Basic et ses fichiers, 1 - méthodes pratiques - Jacques Boisgontier
Le Basic et ses fichiers, 2 - programmes - Jacques Boisgontier
Dictionnaire du Basic IBM - David A. Lien

Turbo Basic
Le livre de Turbo Basic sur PC et compatibles - Jacques Boisgontier
Clefs pour Turbo Basic - Jacques Deconchat

Turbo Pascal - version 3
Turbo Pascal sur PC et compatibles - Frédéric Blanc et Pierre Brandeis
Turbo Pascal pas à pas - Jean-Michel Gaudin
Turbo Pascal et ses fichiers - Jacques Boisgontier et Christophe Donay
Clefs pour Turbo Pascal - Frédéric Blanc et Pierre Brandeis

Turbo Pascal - version 4
Le livre de Turbo Pascal sur PC et compatibles, version 4 - Frédéric Blanc et Pierre Brandeis

Turbo C
Le livre de Turbo C et compatibles - Pierre Brandeis et Francis Pierot
Turbo C pas à pas - Jean-Michel Gaudin
Programmer en C et Turbo C - Claude Nowakowski

C
Clefs pour C - François Piette
Entrées sorties en C - Jean de Brabandt

**Pour connaître nos dernières nouveautés,
ou nous soumettre un problème technique
nous mettons à votre disposition un service minitel**

Service Minitel

Sur le 3615 tapez OI puis * LIV

La loi du 11 mars 1957 n'autorisant, aux termes des alinéas 2 et 3 de l'article 41, d'une part, que les « copies ou reproductions strictement réservées à l'usage privé du copiste et non destinées à une utilisation collective », et, d'autre part, que les analyses et les courtes citations dans un but d'exemple et d'illustration, « toute représentation ou reproduction intégrale, ou partielle, faite sans le consentement de l'auteur ou de ses ayants droit ou ayants cause, est illicite » (alinéa 1er de l'article 40).

Cette représentation ou reproduction, par quelque procédé que ce soit, constituerait donc une contrefaçon sanctionnée par les articles 425 et suivants du Code Pénal.

© Éditions P.S.I., une société de Groupe CEP
6-10, boulevard Jourdan - 75014 PARIS
1988
ISBN 2-86595-489-7

LANGAGES

TURBO PASCAL PAS A PAS

VERSION 4

JEAN-MICHEL GAUDIN

ÉDITIONS P.S.I.
1988

AVERTISSEMENT

Turbo Pascal est un logiciel conçu et commercialisé par **Borland International Inc.,** qui en détient le copyright à l'échelon mondial.

Ce livre n'est pas le manuel du logiciel, c'est un ouvrage complémentaire destiné aux personnes qui veulent se perfectionner à l'utilisation de ce logiciel.

Editions du P.S.I. rappelle que selon la loi n° 85660 du 3 juillet 1985 tout utilisateur de ce logiciel doit être en possession de l'original. Il a toutefois le droit de faire une copie pour ses besoins propres, cette copie n'étant pas cessible à une tierce personne.

Extrait de la loi du 3 juillet 1985

"Toute reproduction autre que l'établissement d'une copie de sauvegarde par l'utilisateur, ainsi que toute utilisation d'un logiciel non expressement autorisée par l'auteur ou ses ayants droit, est passible des sactions prévues par la loi."

SOMMAIRE

Introduction 7

Chapitre 1 : Mise en place... et premier résultat 9

Chapitre 2 : Premier programme 17

Chapitre 3 : Procédurons dans l'unité... 23

Chapitre 4 : Prenons le temps 33
Procédures supplémentaires 37

Chapitre 5 : La fin de REFLEXES ? 41
Est-ce la fin ? 46

Chapitre 6 : Première récapitulation 51
Structure 51
Visibilité 52
Clarté d'un programme 52
Vocabulaire rencontré 54

Chapitre 7 : Le jeu de la vie 59
Notion de tableau 63

Chapitre 8 : La vie, la mort... 71

Chapitre 9 : La fin de la vie ? 79

Chapitre 10 : Passons au graphisme 91
Notion de fonction 93

Chapitre 11 : Franchissons le mur des 64 Ko 99

Chapitre 12 : Deuxième récapitulation 109
Eléments de base du langage 109
Constitution d'un programme 111
Types de variables 112
Opérateurs 115
Instructions élémentaires 115
Procédures et fonctions standards 117

Conclusion 121

Annexe 1 : Préparation des disquettes 123

Annexe 2 : Commandes de l'éditeur 125

Annexe 3 :"Jeu de la vie" 127

Annexe 4 : Codes clavier étendus 133

Index 135

Conseils de lecture 137

INTRODUCTION

Oui, ce livre convient aux débutants en Pascal, qu'ils aient quelques connaissances en informatique ou qu'ils désirent se lancer directement dans ce langage ultra-performant qu'est la version 4. Ceux qui connaissent déjà la version 3 n'auront pas de peine à se glisser dans ce qui va bien au-delà d'une nouvelle présentation : possibilités graphiques importantes, programmes de plus de 64 ko, vitesse de compilation fantastique, routines prédéfinies en très grand nombre, etc.

Non, ce livre n'est pas une version retouchée de **Turbo Pascal pas à pas, version 3.0**, adaptée aux nouvelles possibilités de la version 4.0. Les programmes utilisés ont été totalement repensés, et nous ne parlons même pas de la possibilité de "traduire" un programme de la version 3 vers la version 4.

Mais le principe reste le même : de petits programmes (pas si petits que ça, quand même) pour comprendre progressivement comment programmer en Pascal. En se donnant le plaisir de réaliser sans attendre d'avoir avalé, plus ou moins bien, la liste exhaustive des possibilités et des obligations liées à toute utilisation optimale d'un langage de programmation. Tout le monde n'a pas l'entraînement et le courage nécessaires pour accomplir la performance qui consiste à absorber de sang-froid des centaines de pages avant de commencer à se faire plaisir...

Gros avantage pour ce nouveau **Turbo Pascal 4.0 pas à pas** : il ne s'adresse qu'à une seule catégorie de matériel. Vous utilisez bien sûr un ordinateur "compatible PC". Avec au moins 384 Ko de mémoire et un lecteur de disquettes. Nous n'aurons pas besoin de plus.

Que faut-il connaître pour tirer profit de ce livre ? Bien peu de choses. Savoir allumer un ordinateur, introduire une disquette. Savoir lire est utile, certes... et c'est à peu près tout. Rassurez-vous, vous n'aurez qu'assez peu de frappe à faire, beaucoup moins que le total des listings ! Tout votre travail représente, à peu près, ce que vous trouverez en fin des chapitres 5, 9 et 11.

Où en serez-vous quand vous aurez terminé la lecture de ce livre ? Pas très loin, certes, mais déjà capable d'élaborer des programmes qui fonctionnent (ce qui est la première qualité que l'on demande à un programme), qui soient rapides (deuxième qualité), et qui prouvent que vous êtes capable d'aller plus loin sans difficulté.

Vous aurez utilisé une partie des énormes possibilités graphiques de cette version 4 de Turbo Pascal. Vous aurez aussi dépassé le "mur" des 64 ko, en créant un programme de 75 ko, celui de la fin du chapitre 11. Sans trop avoir à vous fatiguer...

La programmation est un exercice intellectuel qui vaut bien le bridge, les mots croisés, Dallas et le Trivial Pursuit réunis... Du moins est-ce l'avis de l'auteur de ce livre, qui n'en est pas moins capable de goûter à ces plaisirs-là aussi...

Quel plaisir d'apprendre !

CHAPITRE 1

MISE EN PLACE... ET PREMIER RESULTAT

Vous avez entre les mains trois disquettes et un gros manuel. Votre manuel Turbo Pascal 4.0, que vous avez dûment acheté [1]. Le manuel, c'est du papier, pas trop fragile, et pas évident à reproduire quand ça pèse environ 1,2 kg ! Les disquettes, en revanche, sont aussi facile à reproduire qu'à détruire.

Bien sûr, vous n'utilisez pas les disquettes d'origine, mais uniquement des copies. Et même, pour ce livre, des disquettes tirées de vos copies. Tout ce qui suit est prévu pour fonctionner avec un compatible PC "mono-lecteur". Parce que c'est plus simple, et aussi parce que nous n'aurons pas la possibilité d'utiliser tout ce qu'il y a dans vos disquettes.

1. Pirater un produit Borland, avec le rapport qualité/prix qui est le sien, ce n'est plus du vol, c'est de la mesquinerie (déjà dit dans des livres précédents, mais "bis repetita placent", comme disait, euh..., Cléopâtre ?).

Vos disquettes de travail, au nombre de trois, sont identiques. Elles doivent toutes contenir les programmes suivants :

- TURBO.EXE : c'est le programme de Turbo-Pascal 4.0 !
- TURBO.TPL : on en reparlera...

Ils proviennent de la disquette 1.

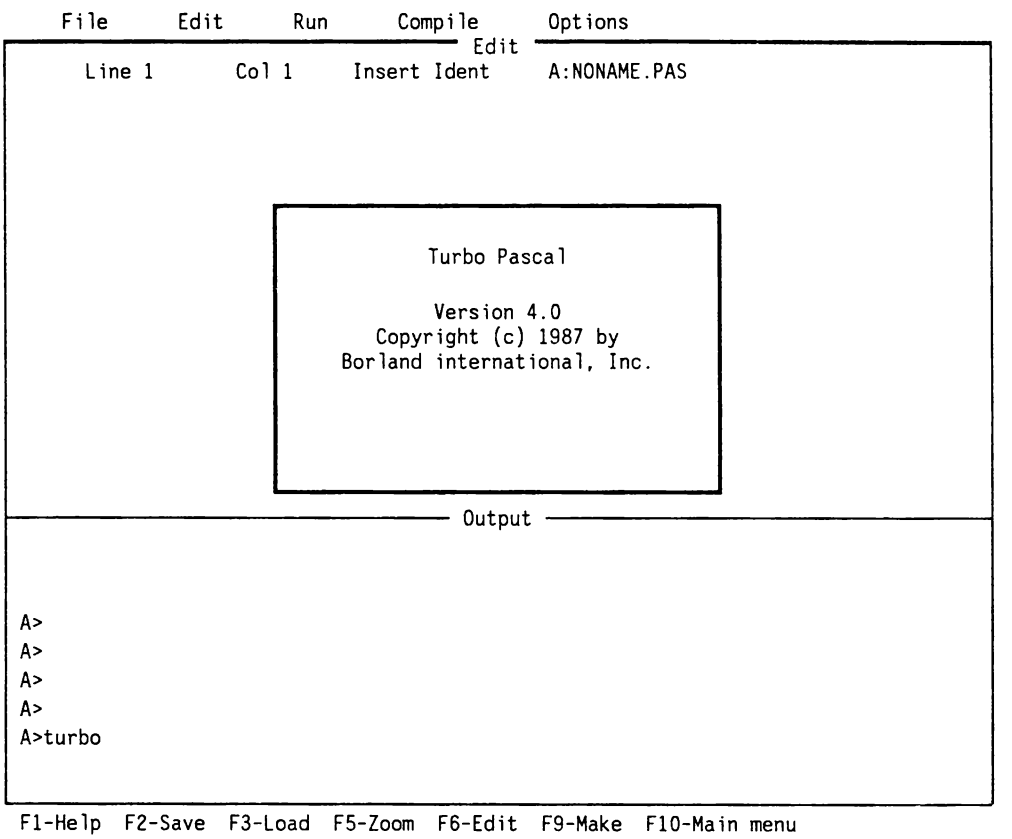
Et aussi, ceux qui viennent de la disquette 3 :

- GRAPH.TPU pour utiliser les graphismes ;
- un des fichiers terminés par .BGI qui correspond à votre moniteur (votre "visu") : ATT, CGA, EGAVGA, HERC, PC3270 ;
- vous devrez aussi installer sur la troisième disquette les quatre fichiers se terminant en .CHR : ils nous permettront de soigner l'affichage (fichiers de "CHaRacters"), à savoir GOTH.BGI (caractères gothiques, mais oui !), LITT.BGI (petits caractères), SANS.BGI et TRIPLEX.BGI pour le style "Sans Serif" et pour tripler les tailles. Mais comme nous aurons bien d'autres choses à faire quand nous serons au chapitre 10, ce n'est pas urgent.

Si vous ne savez pas comment créer ces disquettes, pas de problème et encore moins de panique : l'annexe 1 de ce livre est faite pour vous aider. On vous en dira aussi un peu plus sur ce qu'il ne faut pas oublier, du genre francisation du clavier (KEYBFR). Ah, oui ! Rien ne vous interdit d'utiliser la souris si vous en avez une. Une souris, c'est bien commode pour se déplacer dans les longs listings...

Vous allez numéroter ces disquettes "Turbo 1", "Turbo 2" et "Turbo 3". Sur la première, vous ajoutez le fichier ARTY4.PAS qui se trouve normalement sur la disquette Borland numéro 3. Vous la mettez dans votre lecteur de disquettes. Ouf, nous sommes prêts !

Vous êtes sous MS-DOS, version 2 au moins. Vous devinez qu'il n'y a qu'à lancer TURBO par frappe de TURBO et <Enter>. Et, après quelques bredouillis du lecteur de disquettes, vous voyez apparaître un magnifique écran :



Une pression sur une touche fait disparaître le cadre central.

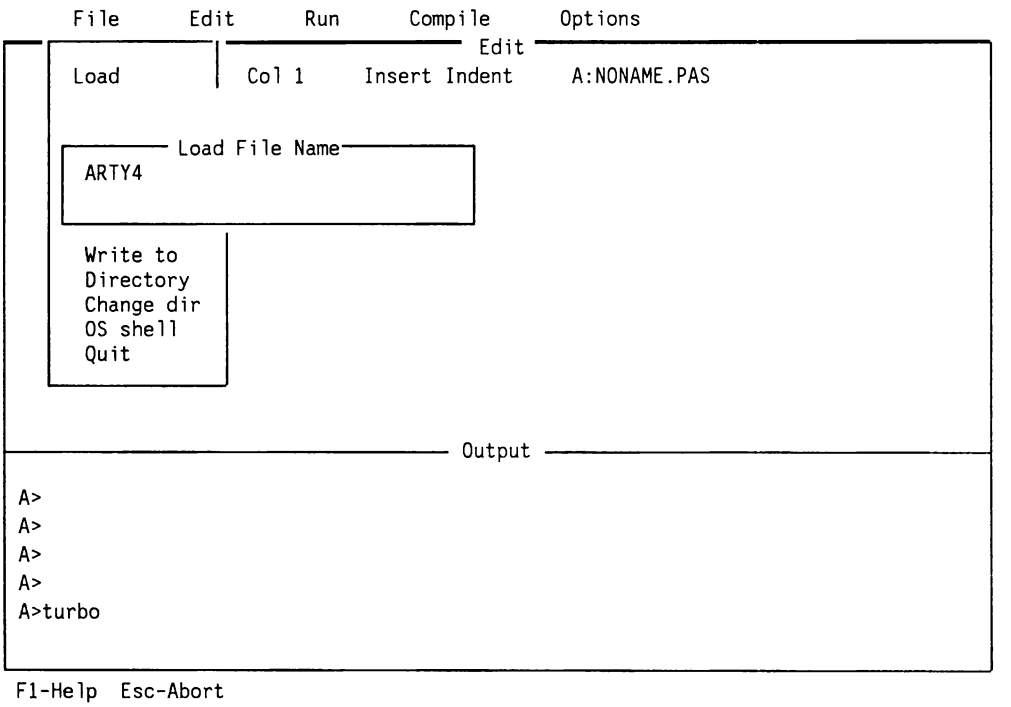
Examinons cet écran :

- la ligne du haut est le menu principal. Elle fera apparaître les menus correspondant aux fichiers (File), à "l'édition" d'un texte (c'est-à-dire au travail sur un texte), au lancement d'un programme, à la compilation (on va y venir dans quelques instants) et à une série d'options bien compliquées pour nous ;
- en haut de la fenêtre supérieure, des indications pour le traitement des textes que sont nos programmes ;
- la fenêtre inférieure nous rappelle ce qui est sorti sur l'écran avant d'être (ou de revenir) sous Turbo (fenêtre des messages) ;
- la ligne inférieure nous indique quelles sont les touches de fonctions utilisables. Par exemple, des jeux dont la subtilité ne vous échappera pas sur

<F5> et <F6> suppriment la fenêtre des messages. Ou la rétablissent si on l'a supprimée...

Comme vous avez sans doute hâte de voir les résultats de Turbo Pascal, nous faisons tout de suite un essai. Nous allons exécuter le programme ARTY4...

Pressez la touche F. Un menu se déroule. Vous comprenez sans peine que l'on peut afficher le répertoire de la disquette (le "directory") dont on ressort par pression sur <Escape>. Ou encore charger un programme. Pressez L (pour Load) ou directement F3, et introduisez le nom ARTY4. Pas besoin de mettre l'extension .PAS : elle est prise automatiquement en cas de défaut d'indication.



Une pression sur "retour-chariot" ou son équivalent, et vous voyez apparaître à l'écran un texte assez long avec plein d'indications bizarres en anglais. C'est le programme ARTY4.PAS...

Pas question pour nous de rentrer dès maintenant dans ses méandres ! Nous allons nous contenter de le faire fonctionner.

Pour l'instant, ce programme est inutilisable par l'ordinateur. C'est un texte, alors que votre PC a besoin d'instructions codées. Le travail du compilateur Turbo Pascal consiste à traduire en langage pour machine, le texte que vous avez sous les yeux.

Vous avez compris, n'est-ce pas, et pensez donc qu'il faut presser la touche C pour compiler ce programme ?

Certes... Mais on peut aller directement à l'exécution en pressant <Alt>-R. Ce que vous allez vous empresser de faire : vous voyez que pendant la compilation, qui ne dure que peu de temps, s'affiche un rectangle avec plein d'indications dedans pour nous dire que tout se passe bien. Et le programme démarre...

Pas mal, non ?

Ce programme donne un si joli résultat que nous allons le mettre sous forme directement exécutable, prêt à l'emploi sans avoir à repasser par Turbo Pascal.

Une pression sur <Escape> ramène dans "l'environnement" Turbo Pascal. Pressez <F10> : on a accès au menu principal (la ligne du haut). Pressez la touche C : un sous-menu apparaît. Faîtes descendre la bande en surbrillance (rougeâtre si vous avez la couleur) sur l'indication " Compilation Memory" et pressez <Enter> : elle devient "Compilation Disk". En remontant la bande sur "Compilation" puis en pressant <Enter>, le programme ARTY4 est mis sous forme compilée sur votre disquette.

Vérifions que le succès clamé par Turbo Pascal est bien là.

Pressez <F10> puis F, puis O. Pas la peine de vous décrire ce qui se passe, vous l'avez sous les yeux : on se retrouve sous MSDOS. Appelez donc le répertoire (DIR) : il est bien là, le programme ARTY4.EXE que vous pouvez, si tel est votre désir, recopier sur une autre disquette et l'offrir à un ami pour qu'il le fasse tourner dans son salon... Notez au passage la différence de taille entre le fichier ARTY4.PAS et son enfant ARTY4.EXE : le premier n'est qu'un texte, le deuxième est un programme exécutable, qui a avalé au passage bon nombre d'octets pré-digérés par TURBO.TPL et GRAPH.TPU.

Revenons sous Turbo Pascal : il suffit de taper EXIT puis presser <Enter>.

Nous allons manipuler quelques instants le système d'édition.

Pour ce faire, nous allons créer un nouveau programme. Vous pensez, bien sûr, au "bouton" : "New", que vous avez vu apparaître dans le bandeau "File". Nous n'utiliserons pas cette possibilité qui a le défaut de ne pas dénommer de

suite le programme sur lequel on travaille. Pressez plutôt L pour "loader" (charger) un fichier comme nous l'avons déja fait pour ARTY4. Bien sûr, vous auriez pû presser directement <F3>.

On vous demande le nom du fichier à charger : mettez ce que vous voulez, en évitant quand même le peu original "ESSAI" que vous risquez de trouver un peu trop souvent, d'où conflits... Prenez par exemple : EDITION.

L'écran se transforme un peu : le cadre supérieur devient vide. Si vous voyez apparaître l'indication "ARTY4 not saved. Saved Y/N ?" c'est que vous avez pressé une ou deux touches pendant que ARTY4 était à l'affichage en mode d'édition, c'est-à-dire que vous avez fait des sottises. Répondez N pour "NO" : laissons-le dans son état d'origine !

Agrandissez le cadre pour bénéficier de la quasi-totalité de l'écran : pression sur <F5>.

Essayez de frapper quelques lignes. Avec n'importe quoi ! Le tout, c'est d'utiliser les principales touches de votre clavier : le "retour-chariot", bien sûr (qui permet aussi d'intercaler des lignes vierges), les touches PgUp, PgDn (qui peuvent avoir un autre nom), les flèches, et la touche de tabulation : cette dernière, comme vous pouvez le constater, décale le curseur depuis le début de ligne jusqu'à la fin du premier mot de la ligne du dessus, ou du suivant si on était déja décalé, etc. Elle nous sera fort utile pour bien présenter nos programmes.

Nous sommes en fait dans un système de traitement de texte. Très proche du bon vieux Wordstar, dont il reprend aussi les commandes essentielles. Si vous êtes un habitué de Wordstar, vous pouvez utiliser ses commandes. En particulier <Ctrl>-Y (un doigt sur la touche <Ctrl>, puis, sans bouger ce premier doigt, un autre sur Y) : effacement de la ligne correspondante ! C'est parfois utile. Le bon vieux <Ctrl>-K-D sert toujours à sortir de l'éditeur, bien qu'ici on puisse le faire plus rapidement par <F10> : si vous avez des habitudes liées à une vieille pratique de Wordstar ou de la version 3 de Turbo Pascal, vous pouvez les conserver.

Bien entendu, les commandes de destruction plus courantes que celles de destruction d'une ligne fonctionnent à merveille. Mais aussi les commandes de "bloc" (morceau de texte) bien utiles, pour aller récupérer dans un programme un morceau qui sert à un autre... Ce n'est pas notre problème pour l'instant. Mais nous avons mis en annexe 2 un récapitulatif des principales commandes type Wordstar, sur une page que vous pourrez découper ou photocopier.

Pour terminer ce premier tour du propriétaire, nous allons quitter Turbo Pascal. On peut le faire de deux façons : passer par le sous-menu "File" en pressant <F10> puis F, puis Q ; ou directement avec <Alt>-X. Dans les

deux cas, Turbo Pascal vous prévient aimablement que EDITION.PAS n'est pas sauvegardé : pas la peine, quand même ! Répondez "N".

Nous avons vu :

- le lancement de Turbo Pascal ;
- la compilation d'un programme ;
- la mise sous forme EXEcutable d'un programme ;
- l'utilisation de l'éditeur.

CHAPITRE 2

PREMIER PROGRAMME

Comme vous le savez fort bien, la qualité des réflexes est, dans notre monde de vitesse, une bonne garantie de survie. Il n'est pas mauvais de connaître ses capacités quand, à l'occasion d'un repas un peu trop arrosé, on se croit au-dessus de tout reproche...

Ne rêvons pas ! Nous n'allons pas fabriquer un véritable réflexomètre. Mais le petit programme que nous constituerons en apprenant Turbo Pascal donnera quand même une précision de l'ordre de cinq centièmes de seconde, ce qui n'est déjà pas si mal.

Le principe est très simple : des "x" s'affichent sur l'écran tant qu'une touche n'est pas pressée. Hélas ! Le vocabulaire de Turbo Pascal est exclusivement en anglais. Si vous avez la chance de savoir que "une touche" se dit "a key", que "presser" se dit "to press", et que "afficher" se dit... Non, rien ne va plus, en fait c'est "écrire" qui est traduit "to write". N'importe, un peu de connaissances en anglo-américain peut faciliter les choses pour se mettre les mémoniques en tête, mais donne parfois des idées complètement fausses, comme nous le verrons !

Le principe de jeu ci-dessus pourrait s'écrire en jargonnant :

" while not a key pressed write 'x' ".

Notre premier programme n'est pas plus compliqué que cela...

Le voici :

```
1)      program reflexe1;   (* premier programme réflexes *)
2)
3)      begin
4)        while not KeyPressed do
5)          Write('x');
6)      end.
```

Examinons-le d'un peu plus près.

Les numéros de lignes à gauche ne sont là que pour le repérage. Le langage Pascal n'utilise pas de numérotation des instructions ! Vous n'aurez pas à les mettre. Vous pourrez tout au plus vous repérer dans un listing avec les numéros de lignes indiqués dans l'éditeur de Turbo Pascal.

Première ligne : elle donne le nom du programme. En Turbo Pascal, elle est facultative. Et pourtant, il vaut mieux la mettre, en donnant si possible au programme le nom que vous utiliserez pour l'enregistrer sur disquette. Si tout le monde faisait ainsi, on n'aurait pas de peine à s'y retrouver quand on échange des listings et des disquettes (expérience vécue...). Le mot "program" est la traduction anglaise de "programme" (mais si, mais si), et c'est un mot du langage Pascal : pas touche ! Le nom du programme, si vous suivez le sage précepte ci-dessus, ne doit pas dépasser huit lettres, comme tout nom de programme sous MSDOS qui se respecte. Ce titre s'achève sur un point-virgule... nous aurons l'occasion d'y revenir très vite !

Toujours pour la première ligne, entre les signes : "(*" et "*)", nous trouvons un commentaire. Très important, les commentaires. C'est là que l'on peut loger tout ce qu'il faut savoir sur le programme : la date de fabrication (elle est certes sur la disquette si l'horloge interne de votre PC est réglée, mais pas sur les listings...), des indications sur ce qu'il fait, comment, pourquoi, et bien sûr l'âge du capitaine.

On peut mettre les commentaires entre accolades, mais ce n'est pas évident à taper avec un clavier gaulois : il faut jouer avec <Alt>-1-2-3 et <Alt>-1-2-5 alors que la touche <Alt> sert aussi à l'éditeur de Turbo Pascal, d'où de joyeux mélanges... On se contentera donc de mettre parenthèse-étoile pour ouvrir un commentaire et étoile-parenthèse pour le fermer, en rêvant au jour où un petit futé aura la bonne idée de mettre un clavier étendu à ces touches sur le marché. Vous direz qu'on peut aussi "reconfigurer" le clavier soi-même, mais ce n'est pas d'une évidence criante.

Vous remarquerez que l'on peut mettre des minuscules accentuées dans un commentaire. Ah, quand même ! Mais il ne faut surtout pas essayer de mettre un commentaire dans un commentaire, sinon le programme de compilation va protester vigoureusement...

Deuxième ligne : Vide. Elle a son importance pour le lecteur. Le compilateur, lui, ne va pas s'apercevoir de sa présence ou si peu que ce n'est pas la peine d'en parler. Quand au programme final, compilé, il ne saura même pas qu'elle a existé. En revanche, c'est très utile de bien séparer les morceaux de programmes en Pascal si on veut pouvoir se relire facilement et être relu de même.

Troisième ligne : les choses se compliquent. Le mot anglais "begin" signifie "commence" en français. C'est donc l'indication, pour le programme, que quelque chose commence...

Quatrième ligne : en jargonnant encore plus que tout à l'heure, c'est "tant-que contraire-de touche-pressée fais".

Il y a, dans cette seule ligne, beaucoup de choses à comprendre !

"While ... do" tout d'abord. C'est du Pascal pur et dur, que tous les langages Pascal possèdent, alors que chacun peut avoir ses petites fantaisies personnelles. C'est : "tant que (condition) faire".

Le "not" signifie "ne pas". Les matheux diraient "non", c'est-à-dire qu'ils disent volontiers "non-bleu" pour dire "pas bleu", par exemple. Vous pouvez penser "contraire de", ce qui est un peu dangereux (il y a beaucoup à dire sur "contraire", "opposé", etc. dans le langage courant). En fait, le "not" agit, opère, sur le résultat de la condition qui suit : si elle est vraie, il donne le même résultat que si elle était fausse et qu'il n'y avait pas de "not", et réciproquement. Bon, cela est dit pour ne pas vous laisser dans l'ignorance, mais ce n'est pas notre problème actuellement !

"KeyPressed" : nous en avons parlé. C'est une "fonction" qui a pour fonction (d'où son nom...) d'indiquer si une touche est pressée ou non. On verra un peu plus tard ce qu'elle fait exactement. Contentons-nous de remarquer qu'on l'a affublée de magnifiques majuscules. Uniquement pour se repérer dans l'écriture : Turbo Pascal confond allègrement majuscules et minucules dans les programmes, ce qui permet d'aller plus vite à la frappe, mais il vaut quand même mieux écrire comme l'explique votre manuel Borland : on s'y retrouve mieux (quoique le manuel Borland use et abuse des caractères gras et de la tabulation de deux caractères, ce qui ne correspond pas du tout à l'éditeur de Turbo Pascal...).

Cinquième ligne : avec un petit décalage de tabulation, nous trouvons un : "Write('x');" qui nous rappelle qu'au bon vieux temps, les ordinateurs servaient surtout à remplir du papier et que l'affichage sur écran était rare. Bon, ce qui importe surtout, c'est de ne pas oublier les apostrophes entre les parenthèses autour de ce qu'on veut écrire (si vous voulez afficher des successions de "O" ou de "zut" rien ne vous l'interdit...), ainsi que le point-virgule de la fin de cette ligne.

Attention à ce point-virgule ! Nous en avons déjà rencontré un au bout de la première ligne (avant les commentaires, certes, mais comme un commentaire compte pour du beurre, on aurait pû le mettre aussi bien après). Un point-virgule est **TOUJOURS** placé à la fin d'une instruction complète, d'une "phrase" qui a un sens en elle-même.

Ne soyez pas inquiet pour l'emploi de ce séparateur. C'est juste une question de bon-sens. Si vous dites "commence" (begin), il y a forcément une suite, donc pas de point-virgule. Si vous dites "tant que", c'est pareil. Si vous dites "affiche telle chose", ça tient debout par soi-même : un point-virgule ! Vous aurez vite assez d'expérience pour ne pas tomber dans les (rares) pièges de ce séparateur.

Sixième ligne (enfin !) : "end."

Ah oui, ce qui signifie "fin", n'est-il pas vrai ? Et, pour la fin, on ne va pas mettre un point "virgule", mais un point final...

Un programme en Pascal commence par "begin" (sans point-virgule !) et se termine par "end." avec un point final... Entre les deux, il peut se passer beaucoup de choses que nous ne vous révélerons que plus tard.

En fait, ce dont vous avez envie, c'est de faire tourner ce programme. Ne perdons pas de temps : allumez votre ordinateur (vous ne l'aviez pas éteint pour lire tranquillement ce qui précède ?). Lancez Turbo Pascal en introduisant "TURBO", et, quand la fenêtre d'annonce de la version est là, pressez directement la touche <F3> pour signifier que vous voulez charger un programme. Répondez REFLEXE1 à la demande du nom du programme (rappelons-le, il est inutile de mettre l'extension .PAS, merci Borland). Vous vous retrouvez dans l'éditeur avec une page vierge qui attend que vous la remplissiez.

Vous allez taper un programme presque identique à celui que nous venons d'analyser. Sans les numéros de ligne, certes. Mais aussi avec une autre différence : nous devons introduire en début de programme une mention "Uses Crt ;" pour signifier à Turbo Pascal version 4 (ce dispositif n'existait pas en version 3) d'utiliser une "unité" qui va lui permettre de comprendre ce que

signifie "KeyPressed". Nous reparlerons de ces unités au début du chapitre suivant : nous en savons assez pour l'instant !

C'est-à-dire que vous devez avoir sur votre écran, dans la partie édition :

```
program REFLEXE1;   (* premier programme jeu de réflexes *)

uses Crt;

begin
     while not KeyPressed do
          Write('x')
end.
```

Il est tapé ? Parfait. Vous voulez l'essayer ? Un peu de prudence ! On pourrait certes le lancer tout de suite par <Alt>-R. Mais ce n'est pas une habitude à prendre. En effet, un programme peut être très convenable vis-à-vis du compilateur, donc se lancer sans aucun problème et puis ensuite "boucler" de façon irrémédiable, sauf à ré-initialiser, voire éteindre l'ordinateur, et alors, adieu à tout ce qui a été tapé ! Il n'y a plus qu'à recommencer...

Prenez de bonnes habitudes : d'abord, on compile en passant par <Alt>-C puis <Return>, ou directement par <Alt>-<F9>. Si le résultat est bon (succès annoncé par Turbo-Pascal), on sauvegarde par pression sur <F2> (touche à bien connaître). Et c'est seulement après cette précaution élémentaire qu'on lance le programme.

Normalement, ici, tout s'est bien passé... Vous avez même sans doute trouvé tout(e) seul(e) qu'il suffisait de presser une touche pour arrêter ce défilement démentiel de "x" ou de "zut" et constaté que l'on revient aussitôt sous Turbo Pascal. Vous avez sans doute aussi essayé de le lancer plusieurs fois de suite pour essayer de diminuer la taille de la partie écrite.

Il est bien certain que ce programme est loin d'être satisfaisant dans l'état où il est !

Patience, cela va venir bientôt !

Nous avons aperçu :

- l'en-tête de programme par *program nom-du-programme* ;
- l'encadrement du corps prinicipal par *begin* et *end.* ;
- l'instruction de structuration *while ... do* ;
- l'instruction *Write( )* ;
- l'opérateur *not* ;
- la fonction *KeyPressed.*

CHAPITRE 3

PROCEDURONS DANS L'UNITE...

Ne dites pas que le programme sur les réflexes tel qu'il est actuellement vous donne satisfaction ! Ce ne sont pas les défauts qui lui manquent.

Nous allons tout d'abord veiller à ce que l'écran soit propre quand le jeu commence. Ce qui nous conduit à utiliser une instruction très classique : ClrScr (pour "Clear Screen" : nettoie l'écran). Mais, de même que "KeyPressed" du chapitre précédent, cet ordre ne fait pas partie intégrante du langage Pascal "standard", celui défini par le Professeur suisse Wirth.

En Turbo Pascal version 4, tout ce qui n'est pas du "grand standard", c'est-à-dire du genre "begin", "end", "write()", est logé dans des fichiers particuliers d'utilitaires : des "unités" que l'on intègre à son propre programme suivant les besoins. En fait, ces unités sont peu nombreuses : il y a ***System***, ***Crt***, ***Dos***, ***Printer***, ***Graph***, ***Turbo3*** et ***Graph3***, c'est-à-dire sept en tout dont les deux dernières servent à utiliser en version 4 des mots de la version 3, ce que nous n'aurons pas à faire ici. ***Crt*** se rapporte à ce qui est "courant", la destination des autres est claire, non ?

Ces "unités" (fichiers inclus) sont chargés automatiquement en même temps que TURBO.EXE. Elles sont dans le fichier TURBO.TPL. Quel avantage, direz-vous ? Eh bien, cela permet entre autres de ne pas surcharger un programme avec des bibliothèques de sous-programmes dont il n'a rien à faire. Nous n'avons pas besoin, nous, de l'imprimante : pas la peine d'ajouter les octets qui correspondent à son mode de fonctionnement... Et puis cela limite les modifications à faire en cas de changement de matériel (on change seulement l'unité ***Dos*** par exemple).

Ce système "d'unités" est très puissant : on peut créer soi-même des "unités" personnelles, qui contiennent de petits sous-programmes utilisés tout le temps. On n'a donc pas besoin de les retaper ni de les mettre dans un listing, ce qui fait qu'ils sont moins chargés, donc plus faciles à lire.

Vous devinez maintenant que notre listing peut se présenter de la façon suivante :

```
program REFLEXE2; (*deuxième programme jeu de réflexes *)

Uses Crt;

begin
     ClrScr;                    (* effaçage de l'écran *)
     while not KeyPressed do
          Write('x');
end.
```

Mais ce n'est pas ainsi que nous allons procéder, c'est le cas de le dire...

On a dû vous apprendre qu'en physique une mesure unique n'est jamais suffisante : il faudra en faire plusieurs. Et à chaque fois, ce sera la même chose, c'est-à-dire nettoyage de l'écran et affichage de 'x'. Ce petit bloc, ce sous-programme, s'appelle en Pascal **une procédure**. Et notre programme peut ainsi être écrit :

```
program REFLEXE2; (*deuxième programme jeu de réflexes *)

Uses Crt;

procedure Joue;
begin
     ClrScr;
     while not KeyPressed do
          Write('x');
end;
```

```
begin
     Joue;
end.
```

On est là en plein dans le vif du sujet : apprendre le Pascal ! Il est vrai qu'on y était déjà, mais du Pascal sans procédure c'est de la bouillie pour les chats...

Vous voyez bien ce qui se passe : le coeur du programme est à la fin, entre le dernier ***begin*** et le ***end.*** de la dernière ligne. Entre les deux, il n'y a que (pour l'instant), l'ordre ***Joue;*** qui renvoie à la procédure du même nom. Et cette procédure pourra (bientôt) être répétée plusieurs fois.

Vous pouvez essayer ce programme. Sans avoir grand chose à taper ! En effet, Turbo Pascal nous donne la possibilité de récupérer un texte (un listing est un texte...) sans difficulté.

Mettez-vous dans le menu File. Vous y voyez "Write to", c'est-à-dire "Ecrire vers". Vous chargez donc, à moins que vous ne l'ayez déjà, le programme REFLEXE1 dans l'éditeur, et, depuis le menu File, vous pressez la touche W et répondez REFLEXE2. La seule modification apparente est que le nom du programme affiché en haut de l'éditeur est passé de REFLEXE1 à REFLEXE2. Il ne vous reste plus qu'à modifier pour obtenir le résultat ci-dessus. Attention aux points-virgules ! Si vous vous souvenez bien de ce qui a été dit au chapitre précédent (toute expression "sensée" est teminée par un point-virgule), vous n'aurez aucune peine à en mettre un où il faut...

Maintenant, essayez-le en prenant soin de compiler d'abord puis de sauvegarder. On n'est jamais trop prudent...

Mis à part l'effaçage de l'écran, il n'y a pas de différence avec le programme précédent. C'est maintenant que nous allons améliorer sérieusement les choses.

Tout d'abord, nous allons répéter dix fois cet affichage. Pour ce faire, nous allons créer un compteur qui partira de zéro pour augmenter de un à chaque fois que la procédure ***Joue ;*** sera "appelée" : l'augmentation du compteur aura lieu à l'intérieur de la procédure ***Joue ;***. On répétera ***Joue ;*** jusqu'à ce que ce compteur soit égal à 10.

En anglais, ***répéte... jusqu'à ce que...*** se dit ***: repeat... until...*** Pas trop difficile, non ? On devine que ce va être quelque chose du genre : ***repeat Joue until compteur=10 ;***

Oui, mais... Nous n'avons pas fabriqué ce compteur, il faut d'abord lui donner un nom. Pourquoi pas *Compteur*, tout simplement ? Il faut aussi préciser à Turbo Pascal de quelle espèce il est, car le langage Pascal ne prend pas de décision qui pourrait être inopportune, du genre "Compteur est un nombre réel", ce que font certains langages, au détriment de la place en mémoire et de

la rapidité d'exécution. Ici, notre compteur varie de 0 à 10, en restant entier ; il n'a pas besoin de prendre comme valeur l'opposé de racine de pi. Bon, c'est un entier, et on peut en faire la déclaration à Turbo Pascal dès le début du programme en écrivant :

var Compteur : integer ;

Ce qui signifie : "Mon cher Turbo Pascal, sache que désormais quand tu rencontreras le terme *Compteur*, ce sera une variable de type nombre entier". Cette variable nous sert dans tout le programme, on dit qu'elle est **globale**.

Le terme de variable vous dérange ? Dites-vous que c'est quelque chose (une boîte, par exemple) qui a un nom et dont le contenu peut varier. Le dit contenu pouvant avoir des caractéristiques différentes (de l'eau, des clous, de l'acide, de l'uranium enrichi), le type de contenant n'est pas le même.

Ce qui est plus gênant, c'est de dire que *Compteur* est un entier : c'est quand même prendre un marteau-pilon pour enfoncer une touche. En effet, un nombre entier, en Turbo Pascal, peut varier entre -32 768 et 32 767, ce qui est beaucoup pour un compteur qu'on destine à une vie entre 0 et 10. Ce n'est pas bien grave quand même, et on en restera là pour l'instant, vous pourrez faire du perfectionnisme plus tard.

Notre programme REFLEXE3 va donc être :

```
program REFLEXE3;  (* troisième programme jeu de réflexes *)

Uses Crt;

var Compteur : integer;

procedure Joue;

begin
     ClrScr;                    (* effaçage de l'écran *)
     Compteur := Compteur + 1;  (* incrémentation compteur *)
     while not KeyPressed do
          Write('x');

end;

begin
     Compteur := 0;             (* initialisation *)
     repeat
           Joue;
     until (Compteur = 10);     (* dix mesures de réflexes *)
end.
```

Il demande encore quelques commentaires.

La déclaration de variable est placée en début de programme. Ce qui est l'habitude en Pascal pour une variable globale. Mais après l'indication d'unité à utiliser, indication qui est une particularité Turbo Pascal version 4, la déclaration de variable est faite à l'aide du signe ":", qui exprime en quelque sorte une indication.

Un peu plus bas, vous voyez écrit :

Compteur := Compteur + 1 ;

qui nous introduit encore du nouveau.

Le signe ":=" signifie "donne à ce qui est à ma gauche la valeur indiquée à droite". C'est ce qu'on appelle aussi le symbole d'affectation. Ici, la nouvelle valeur de ***Compteur*** doit être égale à l'ancienne, augmentée d'une unité. Remarquez qu'on aurait pu employer une autre méthode, en disant à Turbo Pascal de procéder à une "incrémentation" de la variable ***Compteur***, c'est-à-dire en écrivant seulement ***Incr (Compteur) ;*** qui est une instruction Turbo Pascal version 4, plus rapide.

Encore plus bas, on retrouve ce symbole d'affectation dans le "corps" du programme : ***Compteur := 0;*** . Pourquoi ? Parce que *Compteur* n'est pas déjà utilisé, et pourrait être n'importe quel nombre entier. C'est ce qu'on appelle "initialiser une variable".

Encore un nouveau signe après le *until* : cette fois-ci, c'est le bon vieux signe "=" qui intervient. Il est utilisé ici en tant qu'"opérateur relationnel". Nous avons mis ici des parenthèses pour bien faire remarquer la condition que nous imposons ; elles sont, en fait, parfaitement inutiles !

En revanche, ce serait une bonne habitude à prendre que d'écrire non pas "=" mais ">=" : ainsi, on est sûr que le programme s'arrêtera même si, pour une raison obscure, il y a par exemple un passage par 10,00000001 qui n'est pas exactement 10. Dans le cas de notre programme, il est certain que cela n'arrivera pas parce que nous avons pris toutes les bonnes précautions : ***Compteur*** est initialisé à zéro et est entier. Mais supposez que vous ayez oublié d'initialiser et que ***Compteur*** démarre à onze, vous n'êtes pas près de le voir s'arrêter à dix !

Mais pourquoi donc ne pas avoir pris ***while*** au lieu de ***repeat*** ? Très bonne réflexion ! Ces deux instructions sont en fait à peu près équivalentes, sauf que ***repeat*** fait s'exécuter l'intruction qui suit au moins une fois puis vérifie, alors que ***while*** vérifie d'abord. Dans notre programme, on aurait très bien pu utiliser l'une à la place de l'autre (à condition d'intitialiser ***Compteur*** à un, cette fois), ou ne prendre que l'une d'entre-elles... mais quand aurait-on appris l'autre ?

Allez-y, fabriquez ce programme à partir de REFLEXE2 en utilisant "Write to" comme précédemment. Compilation. Sauvegarde. Exécution.

ET IL NE FONCTIONNE PAS CORRECTEMENT ! Il n'y a qu'un affichage, le programme se termine aussitôt après.

Que s'est-il passé ? C'est tout simplement la fonction ***KeyPressed*** qui est en cause. Elle conserve soigneusement la trace du fait qu'une touche a été pressée dès le premier passage. Donc, tout se passe comme si ***Keypressed*** était vrai : pas d'affichage, le programme se termine très vite...

Il ne va pas être bien difficile de remédier à ce défaut. Il suffit d'utiliser, ou plutôt de faire croire au programme qu'on utilise le code de la touche pressée. Il se considérera alors comme dégagé de toute responsabilité, et se mettra en attente d'une nouvelle touche.
(N.B. : ce comportement de ***KeyPressed*** n'est pas le même qu'en version 3).

Nous allons utiliser une nouvelle variable, nommée... ***Touche*** parce qu'elle s'occupe de ces questions de touches, qui servira uniquement dans la procédure ***Joue;*** et qu'il n'est alors pas nécessaire de traîner tout au long du programme : c'est une "variable locale". C'est ce qui est renvoyé par une touche, c'est-à-dire un signe, un "caractère" même si c'est un chiffre. Donc du type ***char*** comme ***Compteur*** était du type ***integer***. Le seul travail que l'on demandera à Touche est de prendre la valeur donnée par la touche pressée au clavier. Ce qui est fait en affectant à ***Touche*** la valeur donnée par la lecture de la touche (to read = lire, key = touche, l'instruction correspondante de Turbo Pascal version 4 est ***ReadKey***) et suffit pour libérer ***Keypressed***...

Notre programme devient alors :

```
program REFLEXE3; (*troisième programme jeu de réflexes *)

Uses Crt;

var Compteur : integer;

procedure Joue;
var Touche : char;
begin
     ClrScr;
     Compteur := Compteur + 1;
     while not KeyPressed do
          Write('x');
     Touche := ReadKey;
end;
```

```
begin
     Compteur := 0;
     repeat
          Joue;
     until (Compteur = 10);
end.
```

Le résultat est encore loin d'être fabuleux. Deux gros défauts : pas de mesure du temps (chapitre suivant) et pas de délai d'attente (on s'en occupe de suite).

Comme la maison Borland a tout prévu pour la satisfaction de son honorable clientèle, nous avons sous la main tout ce qu'il faut pour créer un délai d'attente imprévisible pour le joueur.

Pour avoir un délai, pas de problème : il suffit d'utiliser la procédure toute prête de Turbo Pascal : ***Delay(x)*** où x est un nombre entier. L'ordinateur patientera durant le nombre de milli-secondes (à peu près, dit votre manuel) donné par x. Par exemple, ***Delay(100);*** fera une attente d'un dixième de seconde... à peu près.

C'est un peu plus délicat pour obtenir un délai aléatoire. Mais il existe bien sûr une fonction ad'hoc pour nous donner un nombre aléatoire : c'est la fonction ***Random(w);*** qui nous donne un nombre que nous ne pouvons prévoir, pauvres humains que nous sommes. Suffira-t-il d'écrire ***x := Random(w);*** ?

Pas si vite ! D'abord, il faut savoir ce qu'on indique comme nombre ***w***. Passons sur le fait qu'il est du type ***word***, c'est-à-dire un nombre entier, mais sans signe cette fois, ce qui permet une variation de zéro à 65 535 (pas de panique parce qu'il y a beaucoup de types, on en a déjà vu une bonne partie !). On se contentera de prendre 3 000, c'est-à-dire un temps maximal de trois secondes environ. On laissera un temps fixe de deux dixièmes de secondes d'attente, pour que le malheureux joueur ne se retrouve pas avec une attente nulle, ou presque, à certains moments, ce qui serait d'une méchanceté innommable.

Ensuite, il faut glisser cette indication dans le programme. Et faire en sorte que la séquence d'attente ne soit jamais la même, en initialisant le générateur de nombres aléatoires à l'aide de l'horloge interne de votre PC, ce qui se fait tout seul avec la procédure ***Randomize***.

Notre programme devient :

```
program REFLEXE4; (* quatrième programme jeu de réflexes *)

Uses Crt;

var Compteur : integer;

procedure Joue;      (* affichage après attente aléatoire *)
var Touche : char;   (* récupère le signe renvoyé par KeyPressed *)
begin
     ClrScr;
     Delay(200);            (* pour éviter départ attente nulle *)
     Delay(Random(3000));   (* 3 secondes maxi *)
     Compteur := Compteur + 1;

     while not KeyPressed do
          Write('x');

     Touche := ReadKey;
end;

begin
     Randomize;        (* initialisation *)
     Compteur := 0;    (*   idem         *)

     repeat
          Joue;
     until (Compteur = 10);
end.
```

Vous avez noté qu'on a ajouté quelques remarques, car notre programme commence à prendre un peu de surface : il faut pouvoir s'y retrouver plus facilement par la suite.

Allez-y ! Il est déjà presque tout tapé... si vous reprenez REFLEXE3, procédé dont vous avez bien l'habitude maintenant.

Le résultat commence à ressembler à quelque chose, non ?

Nous avons eu un aperçu de :

- les *Unités* de Turbo Pascal version 4 ;
- la notion de procédure ;
- l'utilisation de *Write to* pour l'éditeur ;
- l'instruction *repeat ... until ...* ;
- déclarations de variables globales ou locales ;
- types de variables : *integer, char, word ;*
- symboles *: :=* = ;
- l'initialisation d'une variable ;
- instructions :
  ClrScr;
  ReadKey
  Delay(...)
  Random(...)
  Randomize

CHAPITRE 4

PRENONS LE TEMPS

Un programme pour mesurer les réflexes qui ne donne pas de mesure, ce n'est pas très sérieux. Il nous faut donc trouver une méthode qui permette d'avoir une mesure du temps à peu près satisfaisante.

Première idée : compter le nombre de 'x' qui sont affichés à l'écran. C'est tout à fait possible, à condition de se fabriquer un étalonnage. C'est même un bon système pour avoir une précision intéressante.

Nous allons faire un peu mieux. Utiliser l'horloge interne de notre PC grâce à une procédure toute prête de Turbo-Pascal. Il s'agit de ***GetTime*** qui nous renvoie l'heure, la minute, la seconde et le centième de seconde de l'instant où elle est requise.

C'est le moment de regarder un peu ce qui est écrit dans le manuel (page 425 du manuel américain : à l'heure où ce livre est écrit, la version française n'est pas encore disponible, mais vous êtes capable de vous retrouver dans une liste alphabétique de procédures et fonctions...).

On vous dit : ***GetTime procedure*** (ce sera peut-être "procédure GetTime" en français...) et, sur la même ligne : ***Dos***, pour exprimer que cette procédure est définie dans l'unité de ce nom. En clair, il va falloir annoncer qu'on utilise l'unité ***Dos***, c'est-à-dire écrire en début de programme un ***Uses Dos;*** de même que nous avions écrit ***Uses Crt;*** dans nos programmes précédents. En pensant qu'il peut y avoir des phénomènes de préséance entre ces unités, quand l'une en utilise une autre (***Graph*** utilise ***Crt***...), ce qui n'est pas le cas ici.

On vous indique ensuite l'utilisation de cette procédure, et, à la ligne suivante, la manière dont elle est déclarée :

GetTime(var Hour, Minute, Second, Sec100 : word)

Ce n'est pas nous qui donnons cette définition : c'est la maison Borland elle-même. On pourrait donc penser qu'il faut l'utiliser telle quelle, avec *Hour* au lieu de *Heure,* etc.

Mais non ! Le langage Pascal est plus souple que cela, et l'on peut dénommer nos variables comme nous l'entendons, au moment où on les déclare (déclaration qui reste obligatoire).

En d'autres termes, avant d'utiliser ***GetTime()***, nous ferons une déclaration de variables "classique" :

var Heure, Minute, Seconde, CentiSeconde : word;

Vous remarquez que nous groupons ces quatre variables dans la même déclaration, et que nous prenons le type ***word*** qui nous est imposé (sinon, conflit de compilation garanti).

Si vous aimez la concision (ce qui est dangereux pour les relectures, mais bien commode quand on n'aime pas titiller le clavier pendant des heures), vous pouvez prendre les notations abrégées, et légales :

var h, min, s, cs : word;

mais, bien franchement, il n'est jamais prudent de prendre des abréviations... bien que, pour des raisons de place sur ce livre, nous ayons pris des notations un peu abrégées pour que tout rentre dans la largeur d'une page !

Cet emploi d'une procédure toute prête est la méthode la plus sûre. C'est celle que nous allons utiliser, en nous disant cependant qu'on pourrait peut-être faire un peu mieux, en serrant de plus près l'horloge interne. On en dira deux mots au chapitre cinq.

C'est dans la procédure ***Joue*** que nous allons faire l'essentiel du travail : déclarer les variables (locales, donc) pour utiliser la procédure ***GetTime()***. Quant au résultat, nous lui donnons le nom de ***TempsRep*** car *TempsReponse* ne permet pas de tout mettre sur une même ligne du livre...

Ne riez pas devant ces mesquineries de dénomination ! Il vaut beaucoup mieux perdre un peu de temps à la frappe et avoir des dénominations claires (et un programme de même) qu'une chose du genre :

uses crt;var c:integer;procedure j;begin clrscr;c:=c+1;while not keypressed do write('x');end;begin c:=0;repeat j;until c=10;end.

qui n'est autre que notre programme REFLEXE3 mis en condensé. Il fonctionne tout aussi bien que celui que nous avons tapé, c'est vrai, à condition d'être sur deux lignes (Turbo Pascal n'accepte pas les lignes de plus de 128 caractères, espaces compris, et il en fait 130). Ce genre de niaiserie fait plaisir à certains, mais avec des fantaisies de ce type on est sûr d'une chose : se planter un jour ou l'autre sans pouvoir s'y retrouver facilement ! Si vous êtes réellement paresseux, vous savez qu'il est bien plus pénible de reprendre un programme que de le taper proprement du premier coup...

Dans notre programme REFLEXE5, nous allons nous contenter de donner le temps de réponse après chaque essai. C'est une phase intermédiaire avant de passer à des choses plus sérieuses. Le principe est très simple : on "pique le temps" (c'est le sens de *get time...*) au début de l'affichage, on en déduit un faux temps de réponse. On déduira le vrai temps de réponse en soustrayant le temps calculé de la même manière lorsque le joueur a pressé une touche. Cette initialisation est un peu hardie : si elle vous pose un problème, vous pouvez définir des variables intermédiaires ***TempsDepart*** et ***TempsFin*** qui donneront le même résultat. Là, ce n'est pas de la simplification abusive que nous faisons, mais du calcul élémentaire...

Comme il est intéressant de voir le fonctionnement du programme, nous avons ajouté trois lignes à titre provisoire : celles qui permettent l'affichage du résultat après chaque pression.

Un ***WriteLn(' ');*** qui ressemble à ***Write();*** et qui en est effectivement très proche : c'est de sauter à la ligne après avoir écrit un espace (ligne = LiNe) pour plus de lisibilité, et surtout un premier aperçu de la manière d'écrire avec une variable. C'est très facile : on écrit entre apostrophes ce qu'on veut voir affiché de son cru, et on met dans l'ordre et à sa place ce qu'on veut voir comme valeur de variable. C'est très logique !

Comme il faut que le joueur ait le temps de lire, on lui accorde une petite seconde de délai...

Le programme devient donc :

```
program REFLEXE5;  (* cinquième programme jeu de réflexes *)

Uses Crt,
     Dos;                          (* "unité" où est GetTime *)

var Compteur : integer;
    TempsRep : word;               (* temps de réponse par essai *)

procedure Joue;
var Touche : char;
    Heure, Min, Sec, CentSec : word;  (* pour GetTime *)
begin
     ClrScr;                       (* effaçage de l'écran *)
     Delay(200);                   (* 3 dixièmes de seconde *)
     Delay(Random(3000));          (* attente aléatoire *)
     Compteur := Compteur + 1;     (* incrémentation compteur *)

     GetTime(Heure, Min, Sec, CentSec);  (* début *)
     TempsRep := CentSec + 100*Sec + 6000*Min + 360000*Heure;

     while not KeyPressed do
          Write('x');

     GetTime(Heure,Min,Sec,CentSec);      (* fin *)
     TempsRep := CentSec+100*Sec+6000*Min+360000*Heure - TempsRep;

     Touche := ReadKey;            (* remise à zéro de KeyPressed *)
     WriteLn(' ');                 (* saut à la ligne suivante *)
     Write('Votre temps de réponse est : ',TempsRep,' centièmes.');
     Delay(1000);                  (* une seconde pour lire *)
end;

begin
     Compteur := 0;                (* initialisation *)
     repeat
          Joue;
     until (Compteur = 10);        (* dix mesures de réflexes *)
end(*REFLEXE5*).
```

Allez-y : il n'y a pas grand'chose à ajouter à REFLEXE4 pour avoir REFLEXE5 ! Vous pouvez bien sûr apporter vos améliorations personnelles, du genre regrouper les *Delay* sous une même parenthèse :

Delay(200+Random(3000));

ou écrire :

Inc(Compteur); plutôt que **Compteur:= Compteur + 1;** ...

L'important est que vous vous y retrouviez plus tard !

PROCEDURES SUPPLEMENTAIRES

Nous allons maintenant organiser notre programme pour qu'il puisse accepter sans peine des améliorations successives.

Il faudrait en effet :

- mettre une introduction, pour prévenir le joueur de ce qu'il aura à faire ;
- effectuer le calcul du temps moyen et donner le résultat en fin de jeu ;
- commenter ce résultat.

Il ne faut **surtout pas** essayer de tout mettre dans la procédure ***Joue*** ! Le programme serait vite illisible...

En fait, nous allons créer des procédures supplémentaires, que nous perfectionnerons au chapitre suivant, en s'appuyant sur le "schéma" :

1) introduction

2) jeu

2bis) totalisation

3) final

Les phases 2) et 2bis) se produisent dix fois. Elles sont liées au *repeat ...until...* du programme principal.

Avec des dénominations "transparentes", le corps (la fin) de notre programme devient donc :

```
begin
     Compteur := 0;
     TempsTotal := 0;

     Introduction;

     repeat
          Joue;
          Totalise;
     until Compteur = 10;

     Final;
end.
```

Les deux difficultés, pas du tout méchantes d'ailleurs, qui conduisent au programme REFLEXE6 ci-après, résultent du travail sur les variables.

D'abord, quoi déclarer ?

Pour le résultat final, il faut faire la moyenne des 10 temps de réponse. On va donner un nombre entier (il serait ridicule de donner des décimales pour un temps de réponse en centièmes de seconde, surtout que la précision que nous obtenons n'est pas fameuse). Du fait de la construction de ***GetTime(),*** nous sommes tenus de prendre le type ***word*** pour les variables du temps. On fait le total des temps de réponse. On divise par 10. Pour l'instant, tout va très bien.

Mais il faut arrondir, et utiliser une des fonctions données par votre manuel. Les deux (*Trunc* et *Round*) donnent un résultat du type ***longint***. Encore un nouveau type ! Cette fois, c'est un entier "long", c'est-à-dire stocké sur quatre octets et non pas deux comme *integer* ou *word*, et qui peut varier de -2 147 483 648 à 2 147 483 647...

Si nous donnons un nom à ce temps moyen, ce qui est préférable pour pouvoir l'utiliser en diverses circonstances, il nous faut bien le déclarer en tant que *longint*. Ne va-t-il pas avoir des problèmes avec les autres, qui sont de petits *integer* ? Et bien non, parce qu'en fait il ne les connaît pas : tout ce qu'il voit, c'est leur moyenne arrondie... Ouf !

Ensuite, où déclarer ?

La variable ***TempsMoyen*** ne sert que dans la procédure ***Final***. On la déclarera donc comme variable locale dans cette procédure. En revanche, ***TempsTotal*** passe de la procédure ***Totalise*** à la procédure ***Final*** : dans l'état actuel de nos connaissances, il nous faut la déclarer comme variable globale, en début de programme.

Ce jeu sur variable locale et variable globale est important. Le langage Pascal est un langage modulaire, et permet d'utiliser des petits morceaux de programmes qui viennent d'ailleurs. A condition que les variables s'accordent bien, ce qui ne peut se faire qu'avec des variables locales et l'usage de fonctions que nous n'avons pas vues pour l'instant. En plus, dans de gros programmes, il peut y avoir des problèmes de "visibilité" d'une variable globale depuis des points trop "éloignés" les uns des autres...

Le listing de REFLEXE6 est ci-dessous. Bien sûr, vous utilisez REFLEXE5 pour le créer. Nous n'aurons plus, dans le chapitre suivant, qu'à perfectionner les différentes procédures sans changer le programme proprement dit. Avantage de la programmation structurée...

Seul point qui peut vous intriguer : la dernière ligne de la procédure *Final* qui est là pour qu'on puisse prendre tout son temps à regarder le résultat. Tant qu'aucune touche n'est pressée, le programme ne fait rien...

```
program REFLEXE6;  (* sixième programme jeu de réflexes *)

Uses Crt,
     Dos;                          (* "unité" où est GetTime *)

var Compteur : integer;
    TempsRep : word;               (* temps de réponse par essai *)
    TempsTotal : word;             (* pour faire la moyenne *)

procedure Joue;
var Touche : char;
    Heure, Min, Sec, CentSec : word;  (* pour GetTime *)
begin
     ClrScr;                       (* effaçage de l'écran *)
     Delay(200);                   (* 3 dixièmes de seconde *)
     Delay(Random(3000));          (* attente aléatoire *)
     Compteur := Compteur + 1;     (* incrémentation compteur *)

     GetTime(Heure, Min, Sec, CentSec);  (* début *)
     TempsRep := CentSec + 100*Sec + 6000*Min + 360000*Heure;

     while not KeyPressed do
          Write('x');

     GetTime(Heure,Min,Sec,CentSec);       (* fin *)
     TempsRep := CentSec+100*Sec+6000*Min+360000*Heure - TempsRep;

     Touche := ReadKey;            (* remise à zéro de KeyPressed *)
     WriteLn(' ');                 (* saut à la ligne suivante *)
     Write('Votre temps de réponse est : ',TempsRep,' centièmes.');
     Delay(1000);                  (* une seconde pour lire *)
end;
(*-------------------------------------------------------------------*)
procedure Totalise;                (* additionne temps de réponse *)
begin
     TempsTotal := TempsTotal + TempsRep;
end;
(*-------------------------------------------------------------------*)
procedure Final;                   (* fin du programme *)
var TempsMoyen : longint;          (* type imposé par Trunc *)
begin
     ClrScr;
     TempsMoyen := Trunc(TempsTotal/10);
     Write('Votre temps moyen : ',TempsMoyen,' centièmes de seconde.');
     while not Keypressed do;      (* !!! provisoire !!! *)
end;
(*===================================================================*)
```

```
begin
     Compteur := 0;              (* initialisation *)
     TempsTotal := 0;            (*      idem      *)

     repeat
          Joue;
          Totalise;
     until (Compteur = 10);      (* dix mesures de réflexes *)

     Final;

end(*REFLEXE6*).
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF*)
```

Nous avons utilisé :

- la procédure *GetTime* de l'unité *Dos* ;
- les types **word** et **longint** ;
- l'instruction *WriteLn()* ;
- la mise en place de procédures supplémentaires ;
- la déclaration de variables locales ou globales ;
- la fonction *Trunc()* ;
- l'écriture de *Write()* avec une variable et du texte.

CHAPITRE 5

LA FIN DE REFLEXES ?

Il nous reste quelques détails à ajouter à notre programme pour qu'il ressemble à quelque chose de sérieux. Et aussi, et surtout, pour continuer à acquérir des notions de Turbo Pascal !

La procédure ***Introduction*** tout d'abord. La pauvrette, elle est vide. Nous allons écrire quelque chose de simple : le nom du jeu, sa date de fabrication, et une ligne d'explications.

Pour que ce soit propre, il ne faut pas écrire n'importe où. Il existe une instruction toute faite pour mettre le curseur où l'on veut : c'est ***GotoXY****(horiz, vertic)*. Le nom des variables est ce que vous voulez ; nous avons mis ici *horiz* et *vertic* pour que vous n'ayez pas de peine à comprendre que... Sachez cependant que *horiz* varie de 1 à 80, et *vertic* de 1 à 25. En d'autres termes, le coin supérieur gauche de l'écran a les coordonnées 1,1 et l'inférieur droit 80,25.

Libre à vous maintenant d'élaborer cette introduction comme vous l'entendez. Celle que nous vous proposons ci-dessous nous permet un premier

aperçu de l'instruction de boucle en faisant une simulation du jeu à l'écran.

Dans un premier temps, le titre du jeu est affiché. Une seconde d'attente (petit truc à connaître : si on ne veut pas que l'utilisateur d'un programme s'ennuie, il ne faut jamais le laisser plus de deux secondes devant un écran fixe). Puis des 'x' apparaissent à partir du coin supérieur gauche de l'écran comme si c'était le jeu, un peu ralenti cependant. Arrêt de cet affichage au milieu de la neuvième ligne. Puis affichage, sous le titre, d'une explication.

Cette procédure nous fait introduire, avons-nous dit, la notion de boucle (une des "instructions répétitives" du langage Pascal). Une boucle est une sorte de compteur, où on introduit une variable (de comptage, dite "l'indice") qui varie depuis un nombre donné jusqu'à un autre. Depuis ... jusqu'à ... se dit *for ... to*. Si *i* est l'indice, la variable de contrôle, et qu'on va de 1 à 10, on écrira : *for i=1 to 10* et ensuite ce qu'il reste à faire.

Ainsin, nous connaissons **trois manières de procéder**, avec *while*, avec *repeat ... until...*, avec *for...to...* ! Ces trois manières ne sont pas tout à fait équivalentes. Nous avons déja vu la différence entre les deux premières. Notez bien que la troisième ne fonctionne qu'avec des nombres entiers, alors que les autres jouent sur des conditions (qui peuvent être liées à des nombres entiers...).

Les boucles ont beaucoup de succès dans les programmes. C'est vrai qu'elles sont faciles à employer. Mais... méfiance ! Il est tentant d'imbriquer des boucles les unes dans les autres, ce que nous ferons d'ailleurs sans vergogne dans le programme sur le jeu de la vie, ce qui complique la lecture.

Dernière remarque sur les boucles : il vaut toujours mieux faire la déclaration de l'indice de contrôle à l'intérieur de la procédure, le prendre donc comme variable locale et non comme variable globale. L'habitude fait qu'on lui réserve en général des petites lettres comme i, j, k... Evitez " l ", qui ressemble fort à "1" dans les listings (nous disons bien "elle" et "un"...).

Voici donc notre procédure :

```
(* procedure Introduction; *)      (* programme INTROREF *)
Uses Crt;       (* pour essai à part *)
                (* cette ligne et les précédentes seront à effacer *)
var i : integer;
begin
     ClrScr;
     GotoXY(30,15); HighVideo; Write('MESUREZ VOS REFLEXES !');
     Delay(1500);
     GotoXY(1,1); NormVideo;
     for i:=0 to 650 do
         begin
```

```
            Delay(3);
            Write('x');
        end;
    Delay(1000);
    GotoXY(10,20);
    Write('Il suffit de presser une touche quand on voit des ''x''.');
    Delay(1500);
    GotoXY(30,15); Write('                          ');
    GotoXY(35,15); TextColor(White+Blink); Write('ATTENTION !');
    Delay(1500); TextColor(White);
end.        (* dans REFLEXES, remplacer le point par un point-virgule *)
```

Nous avons introduit des enjolivures (de petit niveau...) avec ***HighVideo*** (qui met les caractères en surbrillance) et ***NormVideo*** (qui fait le retour à la normale). Il y a aussi ***LowVideo*** (sous-brillance), que nous n'utilisons pas ici. Un peu plus bas, nous avons utilisé la procédure Borland ***TextColor()*** pour faire apparaître un "blink", c'est-à-dire un clignotement. Tout cela est très facile d'emploi !

Cette procédure ***Introduction*** est un programme à elle toute seule. Vous pouvez l'essayer en dehors du programme de base, et l'introduire ensuite dans le programme REFLEXES par le biais des commandes de bloc de l'éditeur (celles de Wordstar). Encore quelque chose de bien utile à connaître : on peut, par ce procédé bien connu des habitués des traitements de textes, prendre dans un listing ce qui est utile à un autre.

Voici comment faire :

- sauvegardez le programme INTROREF ci-dessus ;
- éditez (par <F3>) le programme REFLEXE6 ;
- "piquez" ce programme REFLEX6 pour qu'il devienne REFLEXES, et faites les petites modifications nécessaires (titre en particulier) ;
- placez le curseur dans ce programme REFLEXES entre deux procédures (entre un ***end;*** et un ***procedure***...). Tapez **<Ctrl>-K-R** ;
- Turbo Pascal vous demande le programme à lire : répondez INTROREF ;
- Le programme INTROREF apparaît en sous-brillance (ou sur fond bleu) dans REFLEXES. Tapez **<Ctrl>-K-V** : tout rentre dans l'ordre.

Il ne vous reste plus qu'à faire les petites modifications indiquées (n'oubliez pas de transformer le point en point-virgule) pour que votre programme soit modifié...

Certes, il existe une méthode (par le "Make") en Turbo Pascal version 4, qui permet de faire à peu près la même chose. Mais elle est un peu moins naturelle, et, étant donné la longueur de notre programme, ce n'est vraiment pas la peine d'éviter de tout recompiler !

Modification de la procédure *Totalise* : instruction *if...then...*

Aucune difficulté, mais une modification importante pour le bon déroulement du jeu.

Elle concerne les tricheurs. Si on laisse un doigt traîner sur le clavier, on se voit créditer de quelques temps de réponses égaux à zéro seconde, ce qui arrange quelque peu la moyenne ! La solution est simple : **si** le temps de réponse est inférieur à 6 centièmes, **alors** le joueur se le voit reprocher, et le coup est annulé.

"Si...alors..." se dit en Anglais "if...then...".

Là encore, comme pour *for...to...*, une instruction dont il est bien facile d'abuser ! Il est très tentant d'accumuler les *if...then*, de les imbriquer, et de se retrouver avec un programme aussi lisible qu'un plat de spaghettis trop cuits...

Nous avons employé le "bip" de votre PC : une petite nouveauté, le *Write(#7)* où le signe dièze signifie qu'on emploie le code d'un caractère (drôle de caractère, ici...).

Déjà survolé : l'emploi de la procédure de Turbo Pascal ***TextColor()*** qui utilise des constantes explicites pour les anglophones : si vous voulez afficher en bleu plutôt qu'en blanc, mettez ***Textcolor(Blue)***, et ça marche ! Le mot ***Blink*** signifie "clignotant". En fait, nous utilisons ici des constantes pré-définies dans le Turbo Pascal version 4, qui évitent aux anglophones de mettre en mémoire des séries de codes, ou de feuilleter leur manuel. Répétons que mettre *Blue* (quatre lettres) au lieu de 1 ne change rien à la taille du programme une fois compilé !

La procédure ***Totalise*** peut donc être écrite sous la forme :

```
procedure Totalise;                (* additionne temps de réponse *)
begin
     if TempsRep < 6 then          (* si trop rapide = triche *)
        begin
             GotoXY(35,10); TextColor(White+Blink);
             Write('TRICHEUR !!! '); Write(#7);
             Delay(1000); TextColor(White);
             Compteur := Compteur - 1;     (* annulation essai *)
             TempsRep := 0;
        end;
     TempsTotal := TempsTotal + TempsRep;
end;
```

Modification de la procédure *Final* : instruction *case... of...*

Il s'agit juste de faire des commentaires sur les résultats, en apprenant à utiliser une instruction puissante et claire, bien préférable aux successions de *if...then* que nous critiquions vigoureusement tout à l'heure.

On dira : *suivant les cas de valeurs de la variable ... faire...*

Ce qui s'écrit :

Case (variable) of
valeur_ou_intervalle_1 : instruction1;
valeur_ou_intervalle_2 : instruction2;
etc...
end;

Un bon exemple vaut mieux qu'un long discours !
Voici la nouvelle procédure ***Final :***

```
procedure Final;                    (* fin du programme *)
var TempsMoyen : longint;           (* type imposé par Trunc *)
begin
     ClrScr;
     TempsMoyen := Trunc(TempsTotal/10);
     WriteLn('Temps moyen : ',TempsMoyen,' centièmes de seconde.');
     Case TempsMoyen of             (* commentaires *)
          0..20 : Write('Excellent !');
          21..30 : Write('Convenable.');
          31..40 : Write('Mauvais !');
          40..MaxInt : Write('ALLEZ VITE VOUS COUCHER !');
          end;
     GotoXY(15,15); Write('Une autre partie ?');
     Fin := UpCase(ReadKey);        (* fin si réponse est 'n' ou 'N' *)
end;
```

Dans l'exemple ci-dessus, nous avons pris des intervalles définis par des nombres (MaxInt est l'entier maximum, c'est une constante prédéfinie dans Turbo Pascal), séparés par deux points.

De même que *if...then,* l'instruction *case...of* admet un *else* (autre cas), qui permet de traiter tous les cas pas intéressants. On aurait pû remplacer le 40..MaxInt par ce "else".

N'oubliez pas le *end;* de la fin de *case* !

Deuxième modification : pour éviter de recharger le jeu à chaque fois, il se

déroule tant que le joueur réponds "O" pour "oui" à la question posée. Ou, plutôt, tant qu'il ne répond pas "N" pour "non"... Donc, tant que la variable ***Continue*** n'est pas égale à ***N***, on joue. Il faut donc modifier la fin de la procédure *Totalise* pour que la question soit posée (on retrouve avec plaisir notre vieille connaissance ***ReadKey***, utilisée cette fois à quelque chose de vraiment utile, et on découvre la fonction ***UpCase()*** qui transforme toute minuscule en majuscule).

La partie principale (la fin...) du programme est modifiée pour utiliser cette réponse. Il ne faudra pas oublier de déclarer et d'intialiser la variable ***Continue*** !

```
begin
     Continue := 'O';          (* initialisation *)

     Introduction;

     while Continue <> 'N' do
     begin
          Compteur := 0;       (* initialisation *)
          TempsTotal := 0;     (*    idem        *)

          repeat
                Joue;
                Totalise;
          until (Compteur = 10);

          Final;
     end;
end.
```

Vous ne devriez pas avoir de peine à terminer ce programme en le complétant comme prévu ci-dessus. Nous vous donnons quand même ce programme dans son intégralité en fin de ce chapitre, au cas où vous auriez quelque difficulté de mise en place. Vous remarquerez que nous avons nettement séparé les procédures entre elles : c'est un bon moyen pour s'y retrouver !

EST-CE LA FIN ?

Un programme n'est jamais fini... Celui-ci échappe d'autant moins à cette règle qu'il est bien léger. On peut imaginer d'améliorer la présentation, d'y

mettre de la couleur, de la musique, etc... Plus intéressant serait de mettre les résultats sur imprimante ou sur disquette pour conserver une trace (on verra au chapitre 9 comment procéder).

Plus grave est le fait que la mesure du temps ne donne pas vraiment satisfaction : on se retrouve trop souvent avec les mêmes mesures (22 centièmes est trop fréquent). Il faudrait "creuser" un peu plus, aller toucher directement l'horloge interne et n'en sortir que les centièmes. Nous n'en avons pas ici la possibilité, il faudrait utiliser les "interruptions", ce qui dépasse le cadre de cet ouvrage. Dommage.

Laissons là ce programme. Il nous aura au moins permis de voir bon nombre de choses sur Turbo Pascal. Petite mise au point au chapitre suivant...

Nous avons vu :

- les instructions *case...of* et *if...then...* ;
- l'instruction *for...to...* ;
- les procédures *GotoXY(..,..)* *HighVideo* *NormVideo* ;
- la manière de boucler un programme sans le recharger ;
- l'utilisation du signe # ;
- la fonction *UpCase()*.

```
program REFLEXES;  (* programme jeu de réflexes *)
                   (* TURBO PASCAL V4 PAS A PAS *)
                   (*  J.M. GAUDIN    PSI  1988 *)

Uses Crt,
     Dos;                          (* "unité" où est GetTime *)

var Compteur : integer;
    TempsRep : word;               (* temps de réponse par essai *)
    TempsTotal : word;             (* pour faire la moyenne *)
    Continue : char;               (* caractère pour arrêter jeu *)

procedure Joue;
var Touche : char;
    Heure, Min, Sec, CentSec : word;  (* pour GetTime *)
begin
     ClrScr;                       (* effaçage de l'écran *)
     Delay(200);                   (* 3 dixièmes de seconde *)
     Delay(Random(3000));          (* attente aléatoire *)
     Compteur := Compteur + 1;     (* incrémentation compteur *)
```

```
     GetTime(Heure, Min, Sec, CentSec);  (* début *)
     TempsRep := CentSec + 100*Sec + 6000*Min + 360000*Heure;

     while not KeyPressed do
          Write('x');

     GetTime(Heure,Min,Sec,CentSec);      (* fin *)
     TempsRep := CentSec+100*Sec+6000*Min+360000*Heure - TempsRep;

     Touche := ReadKey;              (* remise à zéro de KeyPressed *)
     WriteLn(' ');                   (* saut à la ligne suivante *)
     Write('Votre temps de réponse est : ',TempsRep,' centièmes.');
     Delay(1000);                    (* une seconde pour lire *)
end;
(*-------------------------------------------------------------------*)
procedure Totalise;                  (* additionne temps de réponse *)
begin
     if TempsRep < 6 then            (* si trop rapide = triche *)
        begin
             GotoXY(35,10); TextColor(White+Blink);
             Write('TRICHEUR !!! '); Write(#7);
             Delay(1000); TextColor(White);
             Compteur := Compteur - 1;    (* annulation essai *)
             TempsRep := 0;
        end;
     TempsTotal := TempsTotal + TempsRep;
end;
(*-------------------------------------------------------------------*)
procedure Final;                     (* fin du programme *)
var TempsMoyen : longint;            (* type imposé par Trunc *)
begin
     ClrScr;
     TempsMoyen := Trunc(TempsTotal/10);
     WriteLn('Temps moyen : ',TempsMoyen,' centièmes de seconde.');
     Case TempsMoyen of              (* commentaires *)
          0..20 : Write('Excellent !');
          21..30 : Write('Convenable.');
          31..40 : Write('Mauvais !');
          40..MaxInt : Write('ALLEZ VITE VOUS COUCHER !');
          end;
     GotoXY(15,15); Write('Une autre partie ?');
     Continue := UpCase(ReadKey);     (* fin si réponse est 'n' ou 'N' *)
end;
(*-------------------------------------------------------------------*)
```

```
procedure Introduction;
var i : integer;                    (* indice de la boucle de démo *)
begin
     ClrScr;
     GotoXY(30,15); HighVideo; Write('MESUREZ VOS REFLEXES !');
     Delay(1500);
     GotoXY(1,1); NormVideo;
     for i:=0 to 650 do
         begin
              Delay(3);             (* petit ralenti *)
              Write('x');
         end;
     Delay(1000);
     GotoXY(10,20);
     Write('Il suffit de presser une touche quand on voit des ''x''.');
     Delay(1500);
     GotoXY(30,15); Write('                          ');
     GotoXY(35,15); TextColor(White+Blink); Write('ATTENTION !');
     Delay(1500); TextColor(White);
end;
(*═══════════════════════════════════════════════════════════════════*)
begin
     Continue := 'O';                   (* initialisation *)
     Introduction;

     while Continue <> 'N' do
     begin
          Compteur := 0;                (* initialisation *)
          TempsTotal := 0;              (*      idem      *)

          repeat
                Joue;
                Totalise;
          until (Compteur = 10);        (* dix mesures de réflexes *)

          Final;
     end;
end(*REFLEXES*).
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF*)
```

CHAPITRE 6

PREMIERE RECAPITULATION

Nous avons déja vu bon nombre de choses. Il est temps d'y mettre un peu d'ordre...

STRUCTURE

Un programme en Pascal se compose de :
- mention (facultative en Turbo Pascal) du nom du programme :
  exemple : *program MONPROG;*
- mention des "unités" employées, particulières à Turbo Pascal 4.0 :
  exemple : *Uses Crt;*
- déclarations de variables. C'est là que l'on indique le nom et le type des variables utilisées tout au long du programme (variables globales) ;
  exemple : *var Touche : char;*
  Compteur : integer;

- procédures, signalées par le mot ***procedure*** ;
- partie principale du programme, encadrée par *begin et end.* ...

Une procédure peut être considérée comme un sous-programme, encadré par *begin* et *end;* ...

Tout ordre (instruction, procédure, fonction) qui a une signification par soi-même se termine par un **point-virgule**.

Exemples :

ClrScr; (effaçage d'écran)
begin (début : il faut une suite !)
Write('Tout va bien à bord.');
if Keypressed then (si une touche est pressée, alors...)

Le vérificateur syntaxique de Turbo Pascal signale clairement les oublis du point-virgule. Ce n'est pas une raison pour les négliger d'office !

VISIBILITE

Une procédure ne peut utiliser des instructions (procédures, fonctions, etc.) que si elles sont définies auparavant dans le programme, en suivant l'ordre de lecture des lignes. Elle ne "voit" que ce qui est au-dessus.

Plus facile à dire qu'à réaliser.

CLARTE D'UN PROGRAMME

Ne pas hésiter à donner des noms explicites (cela ne change rien aux performances du programme une fois compilé).

Exemples :

mauvais : *while not((x-y)>z) do*
bon : *while not((PrixVente - PrixAchat) > Impot) do*

Les ordinateurs actuels ne manquent pas de mémoire. Ne recherchez pas systématiquement les "belles formules" ultra-courtes : il n'est pas évident qu'elles fassent gagner du temps à l'exécution, et il est certain qu'elles sont à

peu près illisibles pour une tierce personne, ou pour l'auteur lui-même au bout de quelques semaines...

Evitez les procédures trop longues. L'idéal est qu'une procédure ne dépasse pas 25 lignes, pour pouvoir tenir dans l'écran.

Une procédure peut contenir ses propres variables, dites variables locales. Diverses raisons doivent vous inciter à utiliser des variables locales plutôt que globales :

- plus grande clarté du programme ;
- facilité de passage de la procédure d'un programme à un autre ;
- moindres risques "d'effets de bord", qui se produisent dans les gros programmes quand il y a trop de variables globales.

En quelque sorte, un programme en Pascal se présente comme suit :

```
program DemoTP4;                (* commentaire *)

Uses unit1, unit2;              (* "unités de TP4 *)

var Truc, Machin : type1;       (* déclaration variables globales *)
    Chose : type2;

procedure PremiereProcedure;
var AutreTruc : type3;          (* déclaration variable locale *)
begin
     Instruction1;
     Instruction2;
     ........
end{PremiereProcedure};

procedure DeuxiemeProcedure;
var AutreChose : Type1;
begin
     InstructionN;
     PremiereProcedure;         (* elle est au-dessus *)
     InstructionTrucmuche;
     if (quelquechose) then
        begin
             InstructionIci;
             InstructionLa;
             DeuxiemeProcedure; (* récursivité... *)
        end;
     InstructionAutre;
end{DeuxiemeProcedure};
```

```
begin                          (* c'est le programme principal ! *)
     Truc := tellevaleur;      (* initialisation *)
     PremiereProcedure;
     DeuxiemeProcedure;
     Instruction;
end{DemoTP4}.
```

Nous avons mis quelques commentaires entre accolades, pour vous rappeler qu'elles existent. L'explicitation de fin de procédure ou de programme comme nous l'avons fait n'a rien d'obligatoire. Cela permet toutefois de mieux se retrouver dans un gros programme.

Il n'y a aucune règle absolue pour les décalages et les tabulations. Même chose pour les espaces : ils ne sont pas indispensables autour, par exemple, de ":=" ou de ";". Mais il vaut mieux en abuser qu'en oublier, sans pour autant en mettre au milieu d'un mot !

Vous avez remarqué que l'une des procédures ci-dessus fait appel à elle-même : c'est la récursivité (le langage Pascal est structuré, déclaratif et récursif...).
Définition du dictionnaire : *Récursivité : voir récursivité.* C'est une particularité très riche en possibilités, que vous aurez plaisir à utiliser (tours de Hanoi, échiquier d'Euler, suite de Fibonacci, etc.).

VOCABULAIRE RENCONTRE

Types standards

char : un caractère, qui peut être défini par son nom (exemple : 'A'), ou son code (exemple : #65). Occupe un octet.

integer : un nombre entier, qui peut aller de -32 768 à 32 767. Occupe deux octets, c'est-à-dire 16 bits.

word : un nombre entier positif, qui peut aller de 0 à 65 535. occupe deux octets, c'est-à-dire 16 bits. Remarquez bien que 2 à la puissance 16, est égal à 65 536, ce qui explique bien des choses pour *word* et *integer*. Du coup, vous comprenez qu'un *char* peut aller de -128 à 127 (2 puissance 8 = 256), et qu'il n'est pas aventureux d'envisager un type qui fasse appel à des nombres positifs sur un octet (octet = byte en Anglais) : bravo, c'est le type **byte**... qui va de 0 à 255, et que vous aurez l'occasion d'utiliser un jour ou l'autre.

longint : entier "long". Sur 4 octets. 2 à la puissance 32, en utilisant les négatifs : de -2 147 483 648 à 2 147 483 647.

C'est tout ce que nous avons vu, et ce n'est déjà pas mal. Bien sûr, il y a des types pour les nombres réels, non entiers, du genre $\sqrt{\pi}$. La question qui se pose est de savoir pourquoi il y a toutes ces catégories. Surtout à une époque où les ordinateurs ne manquent ni de rapidité, ni de mémoire. On pourra faire un essai de vitesses comparées avec divers types pour le jeu de la vie, notre prochaine étape.

Instructions standards

Toutes les variantes du langage Pascal les possèdent.

repeat *instruction* **until** *condition* ;
traduction mot à mot : "répète *instruction* jusqu'à *condition ;*", ce qui n'est pas trop difficile à retenir. Notez qu'on exécute l'instruction au moins une fois.

while *condition* **do** *instruction* ;
mot à mot : "tant que *condition* faire *instruction ;*". Le test est fait en début : l'instruction peut donc ne pas être exécutée.

if *condition* **then** *instruction* ;
mot à mot : "si *condition* alors *instruction;*". Là, ça ne sert qu'une fois. Cette instruction paraît bien facile. Méfiez-vous !

case *variable* **of**
valeur1 : instruction1;
valeur2 : instruction2;
etc.
end;
instruction qui doit remplacer *if...then* toutes les fois où on a plusieurs cas simples : elle est d'une grande clarté dans les programmes. Au lieu de *valeur*, on peut avoir des intervalles, comme nous l'avons vu pour les commentaires du jeu de réflexes.

for *indice=départ* **to** *fin instruction* ;
L'indice sert à compter, c'est-à-dire que l'on fixe au départ le nombre de fois où l'instruction est exécutée. Les boucles sont très appréciées des programmeurs débutants (et des habitués du Basic). N'en abusez pas ! Pensez toujours à *while, repeat,* qui sont plus claires et obligent à plus de rigueur.

Instructions "évoluées"

Ce sont toutes celles qui ont été fabriquées par l'auteur de la version de

Pascal que vous avez entre les mains. En théorie, on peut tout faire avec ce que vous avez ci-dessus, et quelques fantaisies du genre pointeurs... Autant utiliser ce qui est tout prêt, et qui sert souvent.

Faisons la différence entre procédures (elles exécutent un ordre) et fonctions (elles renvoient une valeur de variable, ou des valeurs de variables). Distinction un peu subtile, que le langage C ne connaît d'ailleurs pas : dans tous les cas, on fait naviguer des électrons, non ? Il y a là-dessous des tonnes d'astuces sur le passage par valeur, par paramètre, etc., qui ne nous concernent pas à notre niveau actuel.

Procédures

ClrScr; effaçage d'écran.

Write*(quelquechose)*; sert à écrire sur l'écran, sur l'imprimante aussi comme nous le verrons. L'important, c'est le quelquechose. Différents cas de figure :

- une variable : c'est sa valeur qui est affichée ;
- une suite de caractères (une "chaîne") mise entre guillemets : c'est elle qui est affichée ;
- un mélange des deux : exemple :
  write('x=',x); qui vous donnera, pour x=2 :
  x=2

Petit problème : comment faire pour afficher des apostrophes, puisqu'elles servent d'indicateurs dans *write* ?
Réponse : on en met deux.

Exemple : *write('c''est pas l''amer à boire.');* qui donne :
c'est pas l'amer à boire.

WriteLn(...); : c'est à peu près la même, sauf qu'on passe à la ligne en fin d'écriture *(write line)*.

Delay*(nombre)*; : provoque une attente de *nombre* fois un millième de seconde, à peu de choses près.

Randomize; : initialise le générateur de nombres aléatoires de façon imprévisible pour l'utilisateur. Si on n'utilise pas cette procédure, un programme démarrera toujours avec la même suite. Ici, on se base sur l'horloge interne de votre PC. Difficile de faire mieux.

GetTime*(heure,minute,seconde,centiseconde)*; : renvoie l'heure donnée par l'horloge interne. Renvoie ? Ce ne serait donc pas une procédure, mais une fonction ? Mais non, parce que c'est vous qui fixez les noms des variables. On se contentera de cela... Les variables en question, de type *word*, sont dans l'ordre l'heure, la minute, la seconde, le centième de seconde.

GotoXY*(horiz,vertic)*; : place le curseur à l'endroit indiqué par les coordonnées entre parenthèses. Coin haut gauche : 1,1. Bas droit : 80,25. Bien commode, pas de piège tant qu'on n'utilise pas de fenêtres...

HighVideo; LowVideo; NormVideo; : permettent de faire des enjolivures de présentation (haute, basse, normale intensité).

TextColor*(n)*; : modifie la couleur du texte, suivant la valeur de *n* qui peut aller de 0 à 15 (16 possibilités donc). En prime, le clignotement : on ajoute 128. Pour les anglophones, on peut remplacer ce nombre par un message en clair, ou presque.
Par exemple : *TextColor(LightBlue+Blink);* est équivalent à *TextColor(137)*;.
Qui fera un Turbo Pascal francisé jusqu'au bout des ongles ?
Pour la couleur du fond, employez de même *TextBackground();*

Fonctions : Il nous faut préciser ce qui est donné en retour de leurs emplois.

KeyPressed; : Il faut qu'on en dise deux mots ! Elle renvoie "True" si une touche est pressée, et reste à "False" dans le cas contraire. Nous utiliserons ce type de variable "vrai" ou "faux", le type booléen, du nom de Boole, mathématicien anglais (1815-64) qui a créé la logique algébrique. Dans l'immédiat, il faut bien voir que *KeyPressed* va rester bloquée sur "vrai" tant qu'on ne lui dit rien, c'est-à-dire tant qu'on n'a rien fait du caractère indiqué par la touche pressée. D'où l'usage que nous avons eu de la fonction ci-après :

ReadKey; : renvoie le code de la touche pressée. Donc fonctionne très bien avec *KeyPressed*; ... Ne fait pas d'écho à l'écran. Attend bien patiemment si aucune touche n'est pressée.

Random*(nombre)*; : renvoie un nombre de type *word* au moins égal à zéro et inférieur à *nombre*. Si *nombre* n'est pas indiqué, c'est-à-dire si on écrit simplement quelque chose comme ***x:=Random;*** on se retrouve avec un nombre réel inférieur à 1, qui peut être zéro. Cette fonction est bien commode pour les jeux...

Trunc*(réel)*; : renvoie un entier (de type *word*) qui est la partie entière de *réel.*

UpCase*(caractère)*; : renvoie la majuscule qui correspond à caractère. Que se passe-t-il si *caractère* est une majuscule ? Un chiffre ? Un signe qui n'est pas une lettre ? Rien...

Nous reviendrons sur ces notions en fin d'ouvrage.

EVITEZ DE POURSUIVRE SI VOUS N'AVEZ PAS BIEN COMPRIS CE QUI PRECEDE.

CHAPITRE 7

LE JEU DE LA VIE

Nous commençons à avoir une petite pratique de Turbo Pascal. De là à dire que nous connaissons tout, il y a un fossé que nous ne sauterons pas !

Prenez votre deuxième disquette : à nouveau programme, nouvelle disquette... Nous n'aurons que peu de programmes à écrire dessus, comme vous vous en apercevrez vite.

Nous allons, dans ces chapitres, travailler de façon un peu plus réfléchie que précédemment. Nous apprendrons à utiliser les tableaux, l'imprimante et le lecteur de disquettes, mais aussi et surtout à travailler avec des variables booléennes, ce qui va nous amener à de jolies considérations...

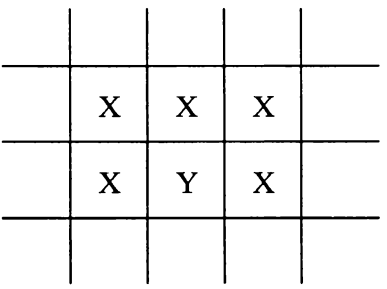
Qu'est-ce que le "jeu de la vie" ? A son propos, certains parlent d'intelligence artificielle. C'est sans doute exagéré, mais cette exagération n'enlève aucun intérêt à ce "jeu".

De quoi s'agit-il ? Une population de cellules étant donnée, avec des règles de reproduction bien établies, on examine ce qui se passe. Les cellules "vivent"

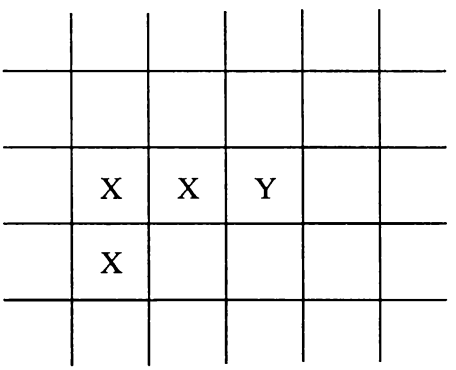
sur les cases d'un quadrillage. Il y a autant de cellules que l'on désire. Leur existence est liée à l'environnement : si une cellule a trop de voisines, ou trop peu, elle meurt. Dans de bonnes conditions de voisinage, il y a reproduction.

Les règles les plus usitées (l'annexe 3 de ce livre vous en dit un peu plus) sont les suivantes :

- si une cellule a plus de trois voisines, elle meurt d'étouffement (de stress?), le voisinage étant pris en diagonale aussi bien que latéralement :



- si une cellule a moins de deux voisines, elle meurt aussi :



En d'autres termes, une cellule existante reste en vie si et seulement si elle a deux ou trois voisines...

- si une case est vide et qu'elle a exactement trois voisines, il y a naissance en cet endroit (quelles moeurs !) :

		X			
		X	X		
		Y	X	X	
				X	X

Pour examiner ces naissances et ces décès, on procéde génération après génération. C'est-à-dire qu'on examine tous les emplacements, vides ou non, du territoire à un instant donné. Par exemple, la petite population ci-dessous :

	X	X	X	

peut-être décrite par :

			n			
		m	x	m		
			n			

m : une seule voisine : meurt
n : case vide avec 3 voisines : naissance

Bien sûr, on peut "jouer" à la main, sur une feuille de papier avec un crayon et une gomme. Mais c'est très vite lassant, surtout si on examine une population un tant soit peu importante. C'est dans des cas comme celui-là qu'un ordinateur montre toute sa puissance.

Nous allons fabriquer un programme qui permet de voir les transformations successives d'une population. Nous n'allons pas chercher à obtenir de super-performances en vitesse d'éxécution : celle que nous atteindrons (plusieurs générations par seconde pour une population simple) est déjà bien suffisante! Ce jeu est un support pour notre découverte de Turbo Pascal, et non pas le but de cet ouvrage. On va quand même faire en sorte qu'il soit présentable...

Surtout, nous allons écrire ce programme de façon beaucoup plus raisonnée que le précédent, qui était, disons-le, de la "bidouille"!

Examinons comment nous allons pouvoir nous organiser. Pour les populations utilisées, d'abord :

- le joueur doit pouvoir introduire la population de son choix ;
- si une population est intéressante, il faut pouvoir conserver au moins sa configuration initiale sur disquette ;
- il serait heureux de pouvoir modifier une population en cours d'évolution (en lui ajoutant ou retranchant des cellules). Pour ne pas alourdir ce livre, nous ne le ferons pas ici, mais ce n'est pas bien difficile avec le traitement que nous employons.

Pour le déroulement, on pourrait proposer trois options :

- un fonctionnement manuel, c'est-à-dire que c'est le joueur qui, par pression d'une touche, commande le passage d'une génération à la suivante. Ce dispositif permet de prendre tout son temps ;
- un fonctionnement automatique, où le changement de générations est effectué un nombre donné de fois. Il est bon que le nombre de générations fixé et le numéro de la génération affichée apparaissent à l'écran ;
- une "décoration" : c'est à peu près le fonctionnement automatique, sauf que le déroulement se poursuit jusqu'à pression sur une touche et que seules les cellules sont affichées.

Il faudrait de plus prévoir une continuation de l'option "automatique", pour pouvoir ajouter des générations à une population dont le développement est spectaculaire sans avoir à repasser par les menus.

Nous ne constituerons pas ces divers "menus" : le lecteur pourra facilement les écrire alors qu'il nous faut de la place pour parler un peu (troisième partie) des graphismes... Il y a quand même un exemple de menu, rassurez-vous.

En ce qui concerne le territoire de vie, nous nous contenterons de l'écran, c'est-à-dire un tableau de 24 lignes (on garde la dernière pour des inscriptions) sur 80 colonnes, ce qui fait un espace vital déjà intéressant... Nous allons donc considérer l'écran comme étant un tableau de 24 cases sur 80.

NOTION DE TABLEAU

Il existe en Pascal un type de variable qui se prête très bien à la description d'un tableau : c'est le type ***Array[...]***.

Grosse différence par rapport aux variables précédentes : il vaut mieux définir le type avant de définir la variable. En effet, deux tableaux peuvent être très différents : entre un tableau de 20 caractères et un tableau de 20 tableaux de 2000 réels, il y a encore moins de rapport qu'entre un chien des Pyrénées et un Chihuahua... Il est plus prudent de bien expliciter avec une définition de type.

Il y a des tableaux très simples : un tableau de 10 entiers, par exemple, que l'on nommera par la déclaration de type suivie de la déclaration de variable :

```
type CeTableau = Array[0..9] of integer;
var MonTableau : CeTableau;
```

On dit ici que le type est déclaré par l'utilisateur. Au cas où vous n'auriez pas oublié, ***Array*** signifie *tableau*. Ici, on a bien des tableaux d'entiers.

Remarquez bien que la déclaration de type est faite à l'aide du signe "=" et non pas " : ". Une fois que la déclaration est faite, le compilateur "reconnaît" ce type, et vous pouvez déclarer ensuite plusieurs variables qui lui correspondent, comme on peut le faire avec le type ***integer*** par exemple...

En Pascal, la notation est très souple : on numérote directement lors de la déclaration. Ci-dessus, ***MonTableau[3]*** est le quatrième élément du tableau *MonTableau*, puisque le premier a pour numéro zéro. On aurait pû aussi bien déclarer :

```
type CeTableau = Array[1..10] of integer;
```

et alors *MonTableau[3]* serait bien le troisième élément de *MonTableau*.

Comme nous l'avons esquissé ci-dessus, on peut faire des tableaux de tableaux.

Deux solutions se présentent à nous pour le "jeu de la vie".

Première possibilité : créer un tableau de 24 tableaux de 80 éléments. C'est la plus simple à concevoir, et cela en déclarant :

```
type Organe = Array[1..24] of Array[1..80] of (variable);
```

puis :

```
var Bestiole : Organe;
```

Il est (presque) évident que *Bestiole[7][13]* correspond à la septième ligne et à la treizième colonne, ce qui facilite bien les choses pour l'affichage.

Mais cette méthode ne nous donne aucune possibilité simple pour examiner ce qui se passe en bordure de l'écran (pour les colonnes d'indice 1 et 80, et les lignes 1 et 24) puisqu'on n'a pas de case voisine à gauche (ou à droite, au dessus, au dessous...). De plus, elle oblige à utiliser deux boucles pour décrire l'écran (une de 1 à 24, l'autre de 1 à 80), ce qui ralentit le déroulement du programme. Vous n'y croyez pas ? Prenez le petit exemple ci-dessous :

```
program lent;
var n, i, j : longint;
begin
     WriteLn('début');
     for n:=1 to 100 do
         for i:=1 to 100 do
             for j:=1 to 100 do;
     WriteLn('fin');
end.
```

```
program rapide;
var n : longint;
begin
     WriteLn('Début');
     for n:=1 to 1000000 do;
     WriteLn('Fin');
end.
```

Dans les deux cas, on ne fait que compter de 1 à un million. Mais, du fait des boucles, le premier programme met environ 40% de temps en plus que le second... Si vous ne le croyez pas, prenez un chrono (une trotteuse suffit) et comparez.

Deuxième possibilité : prendre un tableau de 1920 éléments, c'est-à-dire 24 x 80. La description est moins simple. L'élément numéro 30, par exemple, correspond à la trentième place de la première ligne. Le numéro 243 à la troisième place de la quatrième (et non troisième) ligne. De façon générale, l'élément i correspond à des coordonnées :

$$y = (\text{ partie entière de } (i/80)\ ) + 1$$
$$x = i - (80 * (y - 1)\ )$$

Ce n'est pas très simple à première vue. Mais bien efficace. Deux avantages : une seule boucle (de 1 à 1920) ce qui allège aussi l'écriture, et une semi-vérité sur les bords, dans la mesure où un élément tout à droite de l'écran a pour

successeur un élément tout à gauche mais hélas à la ligne du dessous. On dira qu'on a créé des règles spéciales pour la bordure... A coup sûr, cette distorsion n'interdit pas aux "planeurs" (voir annexe 3) de se déplacer, ni aux populations de se développer. Quand aux lignes haute et basse, un traitement spécial peut résoudre leur problème. On en reparlera.

Cette grave décision - choisir un tableau de 1920 éléments comme variable - étant prise, il faudrait savoir ce qu'on va mettre dedans ! Choix non moins grave à faire entre des entiers, des réels, des caractères, etc. Certes, les ordinateurs actuels ne manquent pas de place, mais il est quand même peu élégant de dévorer de l'espace pour rien. Soyons mesquins, visons au plus petit : le type booléen.

Avantages :

- gain de place : une variable booléenne n'occupe qu'un octet, un entier deux, un réel six, avons-nous dit. On peut ainsi passer de 1920 octets à près de 12 000... ;
- meilleure efficacité pour les calculs (essayez de reprendre ce qui suit avec des entiers et comparez les temps) ;
- bonne cohérence avec notre jeu : on dira "vrai" si une cellule est à la place indiquée, "faux" dans le cas contraire ;
- on continue à apprendre. Et le calcul avec les variables logiques est beaucoup plus simple que d'aucuns l'imaginent.

Nous allons donc déclarer en début de programme un certain nombre de variables (globales, elles serviront dans pratiquement toutes les procédures que nous aurons à élaborer) :
- les coordonnées des cellules : ***CellX*** et ***CellY*** qui, bien entendu, n'ont pas besoin de dépasser 80 et restent positives : type *byte* ;
- les tableaux. Pour créer et conserver trace de la population initiale : on le nommera ***Mémoire***. Pour afficher une population : ***Bestiole***. Pour calculer les changements de cette population : ***Change***. Ces trois tableaux sont du même type : ***Organe***.

Maintenant, il nous faut - enfin ! - nous lancer dans le programme proprement dit.

Première chose à faire : le nettoyage des tableaux avec lesquels on va travailler. Quelques boucles dans une procédure à part pour tout remettre à ***False***, au cas où des traces de programmes précédents encombreraient la mémoire.

Seconde : remplir ces tableaux avec des ***True*** qui représentent les cellules. C'est là un gros morceau.

Nous allons profiter du fait que les touches de curseur utilisent des codifications particulières. Ce sont des "caractères étendus", au même titre que les touches de fonction. Leur pression envoie deux codes, le premier étant zéro. Donc, pas de problème pour les reconnaître, d'autant plus que la fonction ReadKey que vous connaissez bien pour l'avoir déjà rencontrée "avale" les deux codes d'un seul coup. Ouf, ça ne sera pas trop difficile ! Suivant les réponses, on pourra envoyer le curseur vers le haut, le bas, etc. On peut aussi valider une position, et on écrira un 'x' sur l'écran, par pression sur la touche "espace". Ou effacer par pression sur <Del>.

Si ça vous amuse, vous pouvez faire la validation par la touche ***<Ins>***: on vous donne les codes en annexe 4. Encore mieux : valider par ***<Enter>***, et alors tout ou presque marche avec la souris. Comme quoi la souris n'est pas difficile à employer...

Plus précisément : une fois la mise en place des diverses indications faites sur l'écran, on passe à une procédure spéciale pour la saisie. On analyse la réponse : si c'est <espace>, alors on met cet emplacement dans *Memoire*, c'est-à-dire qu'on met la valeur *True* à sa bonne place. On fait aussi revenir le curseur, qui s'en va de suite à droite sans qu'on le lui ait demandé.

Si cette réponse a pour premier code : zéro, c'est qu'il s'agit d'un déplacement ou d'une destruction. Il suffit de modifier les valeurs des coordonnées en conséquence, ou le tableau lui-même, et d'afficher. Les codes nous sont imposés. Pas la peine de les apprendre par coeur, ça traîne dans tous les livres !

Si cette réponse est un 'F', on sortira de la saisie. Dans le cas contraire, on y revient sans la quitter : c'est un tout petit exemple de récursivité.

Comme il est toujours utile de vérifier le bon fonctionnement de ce que l'on fait, nous avons ajouté une procédure spéciale pour ce programme VIE1. Il faudra bien entendu l'enlever au chapitre suivant... Elle se contente d'afficher des '1' aux emplacements où on a mis un 'x', et des '0' partout ailleurs. C'est rassurant.

Petite remarque avant que vous vous lanciez dans la frappe du listing ci-après : ce système de saisie et de déplacement du curseur peut souvent vous servir. Il serait utile de le mettre sous forme d'une procédure très générale, récupérable dans d'autres programmes... Ou, mieux encore, sous forme d'Unité, ce qui est un des gros avantages de Turbo Pascal version 4, mais qui est un tout petit peu trop complexe pour nous.

Nous avons vu :

- le type tableau ;
- l'utilisation des touches du pavé numérique.

```
program SAISVIE;                    (* jeu de la vie : saisie des données *)

Uses Crt;

var CellX, CellY : byte;                              (* coordonnées cellule *)

type Organe = Array[1..1920] of boolean;          (* type d'une population *)
var  Bestiole, Memoire, Change : Organe;            (* tableaux de travail *)

(*-------------------------------------------------------------------------*)
procedure Nettoyage;                  (* nettoyage préalable des tableaux *)
var i : word;
begin
     for i := 1 to 1920 do                      (* 1920 = 24 fois 80 ... *)
         begin
              Bestiole[i] := False;           (* remise à zéro population  *)
              Memoire[i] := False;            (*    "      " mémorisation *)
              Change[i] := False;             (*    "      " utilitaire   *)
         end;
end;

(*-------------------------------------------------------------------------*)
```

```
procedure Saisie;                          (* saisie des positions initiales *)
var Reponse : char;
begin
     Reponse := UpCase(ReadKey);             (* trois possibilités utiles *)

     if Reponse = ' ' then begin           (* validation d'un emplacement *)
        Write('X');                                        (* affichage *)
        GotoXY(CellX,CellY);                             (* retour curseur *)
        Memoire[80*(CellY-1) + CellX] := True;          (* mise en mémoire *)
     end;

     if Reponse = #0 then begin  (* c'est une touche "caractère étendu" *)
        Reponse := ReadKey;       (* analyse du deuxième code *)
        Case Reponse of
             #72 : CellY := CellY - 1;              (* haut    *)
             #80 : CellY := CellY + 1;              (* bas     *)
             #75 : CellX := CellX - 1;              (* gauche  *)
             #77 : CellX := CellX + 1;              (* droite  *)
             #83 : begin                            (* annule  *)
                        Memoire[CellX + 80*CellY] := False;
                        Write(' ');
                   end;
             else Saisie;                  (* mauvaise touche... *)
         end;
         GotoXY(CellX,CellY);                        (* déplacement *)
     end;

     if Reponse<>'F' then Saisie;          (* fausse touche ...  *)
end;

(*-------------------------------------------------------------------------*)
procedure MiseEnPlace;                                (* écran de saisie *)
begin
     ClrScr;
     GotoXY(12,25);
     HighVideo;
     Write('Validation : <espace>     Effacement : <Del>      Fin : F');
     NormVideo;
     CellX := 40; CellY := 12;                          (* centre écran *)
     GotoXY(CellX,CellY);
     Saisie;
end;

(*-------------------------------------------------------------------------*)
```

```
procedure Verification;          (* vérification pour CE programme ... *)
var i : word;                    (* A EFFACER TOTALEMENT ENSUITE !!!   *)
begin
     ClrScr;
     GotoXY(1,1);
     for i := 1 to 1920 do
         if Memoire[i] = True then Write('1')

            else Write('0');
end;

(*=============================================================*)
begin
     Nettoyage;
     MiseEnPlace;
     Verification;
end.
(*FFFF
```

CHAPITRE 8

LA VIE, LA MORT...

Il serait temps de se mettre à élaborer ce qui est la partie essentielle de notre programme de "Jeu de la Vie", à savoir celle qui concerne l'évolution d'une population...

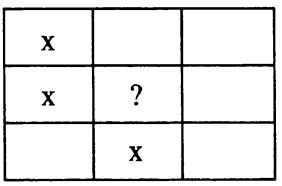
Il nous faut raisonner un peu avant de commencer.

Nous pourrions nous limiter à une "zone de travail" qui n'opère que sur une partie de l'écran, celle où se situe la population : cela réduirait les calculs, et accélérerait le déroulement (quoique l'étude de l'évolution de cette zone prenne elle-même du temps). Ce n'est pas nécessaire en Pascal 4, qui est suffisamment rapide pour nous permettre de travailler sur tout l'écran.

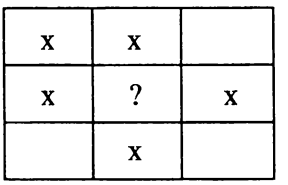
C'est alors assez simple.

Examinons la situation en un point du "domaine" des cellules (l'écran) :

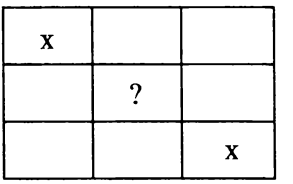
- **Premier cas :** pas de cellule en place. Pour qu'il y ait naissance, il faut qu'il y ait exactement trois cellules dans les huit cases environnantes. Autre façon de l'exprimer : changement de situation à cet endroit si trois cellules au total ;



naissance

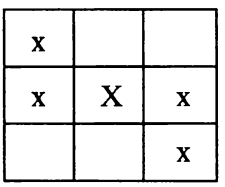


rien

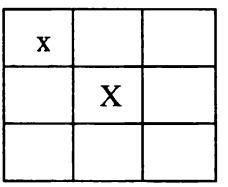


rien

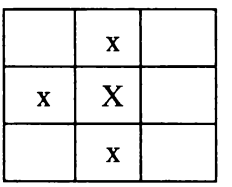
- **Deuxième cas** : une cellule en place. Il y a décès s'il y a plus de trois ou moins de deux cellules autour. Autrement dit, changement à cet endroit si moins de trois ou plus de quatre cellules au total (celles en place + celles autour).



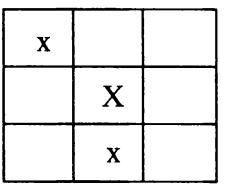
décès



décès



rien



rien

Dans les deux cas, donc dans tous les cas, il va nous falloir compter combien il y a de cellules dans les neuf cases qui constituent le secteur d'étude. Nous appelerons ***Karac*** ce nombre qui **caract**érise le point. De quel type est ***Karac*** ? Il est au plus égal à 9, donc un *byte* est bien suffisant ! Nous déclarerons uniquement **Karac** dans notre fonction de calcul : cette variable ne sert que dans cette dernière, inutile de surcharger nos déclarations du début.

Nous allons donc explorer le tableau de 1920 cases que forme l'écran en comptant pour chacune le nombre de cases occupées dans la "zone utile", c'est-à-dire en calculant **Karac**. Si une case a la position **i**, celle au-dessus a la position **i - 80** (il y a 80 caractères par ligne), celle au-dessous la position **i+80.**

Mais... les variables des tableaux sont des Booléens, que l'on ne peut pas additionner, sauf en passer par la fonction *Ord()* qui nous renvoie 1 si on a *True* et 0 si on a *False*.

C'est-à-dire que **Karac**, une fois le tableau *Bestiole* copié sur *Memoire*, pourrait être donné par :

```
Karac = Ord(Bestiole[i-81]) + Ord(Bestiole[i-80]) + Ord(Bestiole[i-79])
      + Ord(Bestiole[i-1]) + Ord(Bestiole[i]) + Ord(Bestiole[i+1])
      + Ord(Bestiole[i+79])+Ord(Bestiole[i+80])+ Ord(Bestiole[i+81]);
```

Ouf ! C'est une écriture bien lourde, qui demande a être allégée. De toute façon, Turbo Pascal la refuse (ligne trop longue) !

L'idée première est de mettre une petite boucle pour faire les calculs par paquets de trois (l'indice de la boucle variant de -1 à +1, ou de -80 à +80 avec un pas de 80). C'est même très tentant. Vous pourrez essayer : le fait de prendre une boucle va... ralentir très nettement le déroulement du jeu (de l'ordre de 80% plus lent !). En effet, le programme est alors tenu de vérifier l'évolution de la boucle pendant ce calcul, d'où une perte de temps. Les solutions les plus "belles" ne sont pas forcément les meilleures !

Nous allons prendre pour ce chapitre une solution intermédiaire, qui consiste à ne faire qu'une boucle en utilisant un tableau des valeurs ci-dessus. C'est uniquement pour travailler un peu plus sur les tableaux! Il vaudrait mieux utiliser un autre type, le type *énumération* qui, comme son nom l'indique, est prévu pour les énumérations. On ne peut quand même pas tout faire...

Ce tableau auxiliaire, *Environ*, qu'on va définir globalement pour se simplifier la vie (nous ne le conserverons pas dans le listing du chapitre 9), nous le définirons par :

```
type Autour = Array[1..9] of integer;
var Environ : Autour;
```

Nous donnerons aux éléments de *Environ* leurs valeurs en début de Programme (-81,..., 79, etc.).

C'est grâce à lui que nous ferons l'examen de l'emplacement *i* pour calculer Karac. Avec une boucle :

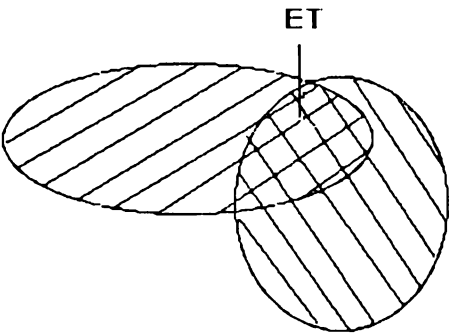
```
for j := 1 to 9 do
    if Bestiole[i+Environ[j]]=True then Karac:=Karac+1;
```

Remarquez que nous pouvons nous dispenser d'écrire *=True* : on est déjà dans les booléens. Ce que nous avons mis ci-dessus est un peu lourd : on écrit, **"si Bestiole[etc.] est égale à vrai, c'est vrai"...**

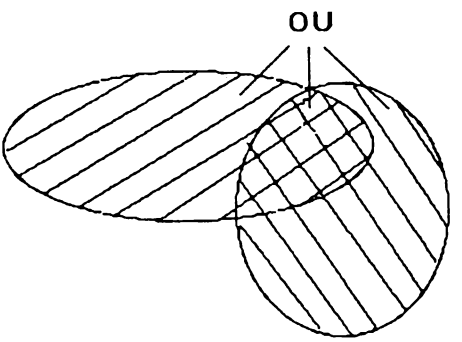
C'est maintenant que les choses se corsent ! Il va nous falloir raisonner sur les booléens.

Petit rappel de mathématiques ô combien élémentaire. Le "et" et les "ou". On a bien dit LES "ou". Vous vous souvenez sans doute avoir vu des "patates" qui étaient très à la mode dans l'enseignement (elles ne le sont plus, sans doute parce que c'était trop commode).

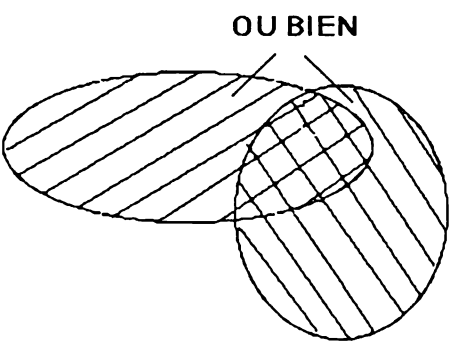
Si on prend ce qui concerne les deux patates en même temps, on a le "ET" logique, en anglais AND :



Si on prend tout ce qui concerne au moins une patate, on a le "OU" logique, le "ou inclusif", qui est le "ou" des mathématiciens et pas tout à fait celui de la pratique courante. Dans l'exemple ci-dessous, si la patate de gauche représente les fumeurs de pipe, celle de droite les blonds, le "OU" fera considérer les blonds qui ne fument pas la pipe, les bruns non-fumeurs et aussi les blonds fumeurs de pipe. En anglais : OR.



Mais il est souvent intéressant de ne prendre que ce qui a une seule des deux "qualités", ce qui est l'un ou bien l'autre. C'est le "ou exclusif", qui correspond souvent au langage courant. Les blonds fumeurs de pipe ne sont plus dedans. En anglais informatique : XOR.



Utilisons ces notions pour notre problème.

D'après ce que nous avons dit plus haut, il n'y a pas de changement si une cellule en place est entourée de trois voisines, ou si une place vide est entourée de quatre cellules. En d'autre termes, il est nécessaire que *Karac* soit différent de 4 pour qu'il y ait changement : on ne travaillera que dans ce cas qu'en écrivant :

if NOT(Karac=4)

écriture que nous préférons à :

if (Karac<>4) qui est pourtant équivalente, mais le ***NOT*** se voit très bien !

Examinons alors les cas possibles :

	Karac = 3	NOT(Karac=3)
Bestiole[i] est vrai	rien ne se passe	décès
Bestiole[i] est faux	naissance	rien ne se passe

que l'on peut aussi écrire :

	Karac = 3	NOT(Karac=3)
Bestiole[i]=True	Change[i]=False	Change[i]=True
Bestiole[i]=False	Change[i]=True	Change[i]=False

C'est-à-dire que, toujours dans le cas où Karac est différent de 4, *Change[i]* ne devient *True* que lorsque *Bestiole[i]* est *True* **ou bien** lorsque *Karac=3.* Ou encore, ce qui revient au même, *Bestiole[i]=False* **ou bien** *Karac<>3.* Prenons la première formulation. Elle peut s'écrire :

```
if NOT(Karac=4) then
   if (Bestiole[i] XOR (Karac=3)) then
      Change[i]:=True;
```

On peut faire encore mieux ! N'oubliez pas que les *if...then...*sont gourmands en temps d'exécution.

Nous avons dit que *Change[i]* devient *True* quand *(Bestiole[i] XOR (Karac=3))* est *False*, n'est-ce pas ? Autrement dit, *Change[i]* prend la valeur de *(Bestiole[i] XOR (Karac=3))* ce qui nous permet d'écrire :

```
if NOT(Karac=4) then
   Change[i] := Bestiole[i] XOR (Karac=3);
```

Pourquoi s'arrêter en si bon chemin ? *Change[i]* ne devient *True* que lorsque *NOT(Karac=4)* l'est aussi, et cela en même temps que *(Bestiole[i] XOR (Karac=3))*, non ? On peut donc écrire :

```
Change[i] := NOT(Karac=4) AND ((Bestiole[i] XOR (Karac=3));
```

Nous n'avons plus qu'une seule ligne de calcul, et nous avons éliminé les *if...then* ! C'est là une jolie écriture, rapide dans son fonctionnement, mais qui présente le défaut de ne pas être d'une clarté lumineuse, c'est le moins que l'on puisse dire. Si vous vous sentez perdu(e), vous pouvez employer une série d'instructions plus explicites, dont les onze lignes remplacent celle ci-dessus, mais avec pour contrepartie un sérieux ralentissement :

```
if Karac <> 4 then
if (Bestiole[i] = True) AND (Karac <> 3) then
   Change[i] := True;
if (Bestiole[i] = False) AND (Karac = 3) then
   Change[i] := True;
if (Bestiole[i] = True) AND (Karac =3) then
   Change[i] := False;
if (Bestiole[i] = False) AND (Karac <> 3) then
   Change[i] := False;
if Karac = 4 then
   Change[i] := False;
```

Ne pensez pas qu'on peut éviter les mises à *False de Change[i] :* si vous ne le faites pas ci-dessus, il faudra bien le faire dans une procédure de remise en ordre, sinon vous allez traîner des scories d'études d'une population précédente d'où des résultats aberrants.

Notre procédure ***EtatCivil*** devient donc, en ce qui concerne l'analyse et la mise à jour du tableau ***Change*** :

```
procedure EtatCivil;                    (* fait évoluer une population *)
var i,j : word;                          (* indices pour les boucles *)
Karac : byte;                      (* caractérise environs cellule *)
begin
    GotoXY(1,25); Write(Compteur); Inc(Compteur);
    GotoXY(12,25);
    Write('          Pour finir : presser une touche.              ');
```

```
for i := 81 to 1839 do    (* on évite première et dernière ligne *)
    begin
        Karac := 0;
        for j := 1 to 9 do                (* examen des environs *)
            if Bestiole[i+Environ[j]] then Karac := Karac + 1;
        Change[i] := NOT(Karac=4) AND (Bestiole[i] XOR (Karac=3));
     end;
```

Elle suppose bien sûr que le tableau ***Environ*** ait été déclaré et initialisé par ailleurs.

Reste à procéder à l'affichage. Nous le ferons, avons-nous dit, dans la même procédure. Avec une boucle de 1 à 1920, qui est simple à écrire. S'il y a changement, on procède à un certain nombre d'opérations :

- changement de *Bestiole[i]* ;
- remise à *False* de *Change[i]* pour être prêt à étudier la génération suivante ;
- affichage d'un 'x' (en fait, nous avons mis la lettre grecque thêta qui fait plus joli...) ou d'un blanc suivant les cas.

Rien ne vous interdit de procéder à des variantes, en vous armant d'une montre avec trotteuse et en essayant toujours la même population. Par exemple en remplaçant les *for... to* par des *while* (avec un initialisation préalable de l'indice, et l'emploi de *Inc(i)...)*.

Dernière remarque : nous avons utilisé *Trunc()* (pour obtenir la partie entière) en transmettant la valeur retournée à *CellY* qui a été déclarée comme étant une variable de type *byte*. Or, la fonction *Trunc* est censée retourner une variable de type *longint...* qui, comme chacun sait, est beaucoup plus grosse. Ce n'est pas bien du tout ! Mais comme on est sûr que le résultat est inférieur à 255 (limite de *byte*), ça passe quand même, mais ce n'est pas une habitude à prendre.

Vous avez ci-après le listing qui correspond à la fin du programme VIE. Nous n'avons pas mis le début : mis à part le changement de titre et la déclaration du tableau *Environ* et celle du *Compteur* (de type longint) en début de programme, il vous suffit de reprendre le listing de SAISVIE par "Write to", d'en supprimer la fin, et de taper ce qui suit.

Vous avez maintenant en main un "jeu de la vie" qui fonctionne, que vous pouvez essayer, et que nous allons nous empresser d'améliorer.

Nous avons vu :

- les opérateurs logiques AND, OR et XOR ;
- leur emploi à la place de *if...then* ;
- un algorithme pour le jeu de la vie.

```
(*-------------------------------------------------------------------------*)
procedure Initialise;          (* initialisation tableau  bestiole ... *)
var i : word;
begin
     Compteur := 1;                          (* ... et "âge" population. *)
     for i:=1 to 1920 do Bestiole[i]:=Memoire[i];
end;

(*-------------------------------------------------------------------------*)
procedure EtatCivil;                    (* fait évoluer une population *)
var i,j : word;                        (* indices pour les boucles *)
Karac : byte;                        (* caractérise environs cellule *)
begin
     GotoXY(1,25); Write(Compteur); Inc(Compteur);
     GotoXY(12,25);
     Write('            Pour finir : presser une touche.              ');
     for i:=81 to 1839 do    (* on évite première et dernière ligne *)
         begin
              Karac := 0;
              for j := 1 to 9 do                (* examen des environs *)
                  if Bestiole[i+Environ[j]] then Karac := Karac + 1;
              Change[i] := NOT(Karac=4) AND (Bestiole[i] XOR (Karac=3));
          end;

     for i := 1 to 1920 do      (* mise à jour tableaux  -  affichage *)
         if Change[i] then begin
            Bestiole[i] := NOT Bestiole[i];
            Change[i] := False;      (* au passage : remise à False *)
            CellY := 1 + Trunc(i/80);
            CellX := i - 80 * (CellY-1);
            GotoXY(CellX,CellY);
            if Bestiole[i] then Write(#233) else Write(' ');
            end;
end;

(*=========================================================================*)
begin
     Environ[1]:=-81; Environ[2]:=-80; Environ[3]:=-79;
     Environ[4]:=-1; Environ[5]:=0; Environ[6]:=1;
     Environ[7]:=79; Environ[8]:=80; Environ[9]:=81;
     Nettoyage;
     MiseEnPlace;
     Initialise;
     while not KeyPressed do EtatCivil;
end.
(*FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF*)
```

CHAPITRE 9

LA FIN DE LA VIE ?

Nous allons perfectionner le programme VIE avec trois objectifs :

- améliorer la vitesse d'exécution ;
- mettre en place un "menu" pour l'utilisateur ;
- surtout, puisque ce livre est fait pour apprendre, découvrir comment utiliser le lecteur de disquettes et, du même coup, comment se servir de l'imprimante.

Améliorer la vitesse d'exécution ne présente aucune difficulté. Nous savons qu'il suffit de réduire le nombre de boucles et de *if...then*, ce que nous allons faire : nous prenons notre courage à deux mains et tapons toutes les valeurs possibles de la "coordonnée" de *Bestiole* quand on veut calculer *Karac*. Nous utilisons la fonction *Ord()* qui nous retourne : **1** si *Bestiole[i]* est vrai, **0** dans le cas contraire. Ne croyez pas que le fait que *Karac* se soit transformé en un simple K soit dû à une quelconque paresse de notre part, c'est tout simplement pour une question de mise en page pour ce livre...

Notre procédure *EtatCivil* devient alors, du moins son début :

```
procedure EtatCivil;                    (* fait évoluer une population *)
var i : word;                          (* indice pour la boucle *)
K : byte;                         (* caractérise environs cellule *)
begin
  GotoXY(1,25); Write(Compteur); Inc(Compteur);
  for i:=81 to 1839 do begin   (* on évite première et dernière ligne *)
    K:=Ord(Bestiole[i])+Ord(Bestiole[i-80])+Ord(Bestiole[i+80]);
    K:=K+Ord(Bestiole[i-1])+Ord(Bestiole[i-81])+Ord(Bestiole[i+79]);
    K:=K+Ord(Bestiole[i+1])+Ord(bestiole[i-79])+Ord(Bestiole[i+81]);
    Change[i]:=NOT(K=4) AND (Bestiole[i] XOR (K=3));
  end;
```

Le résultat est tout spectaculaire : la vitesse est pratiquement doublée ! Autre avantage : on peut, de façon relativement claire, faire en sorte que la première et la dernière ligne ne soient pas tenues à l'écart ; on fait une petite boucle (de 1 à 80) pour la première en répétant le calcul de Karac avec les valeurs qui font intervenir la dernière. Pas de ralentissement sensible. En revanche, si vous voulez que les mauvais effets sur les frontières verticales disparaissent, il y a davantage d'intructions à écrire... et une exécution ralentie.

Passons à un peu plus difficile : l'usage du lecteur de disquettes. Pour commencer, on va parler de l'imprimante... Il y a en effet un rapport !

Nous voulons écrire avec l'imprimante. Il suffit d'utiliser ce bon vieux *Write*, mais avec une mention complémentaire : l'indication du "canal" que l'on utilise, à savoir ici l'imprimante.

Exemple : ***Write(Lst,'Bonjour');*** vous envoie "Bonjour" sur l'imprimante. Tout se passe comme si *Write* attendait toujours l'indication du canal sur lequel écrire, et prenait l'écran si on ne lui dit rien.

L'indication ***Lst*** fait partie du vocabulaire Turbo Pascal, ce n'est pas nous qui définissons ce mot. De plus, en version 4, il faut préciser que l'on emploie l'unité ***Printer***.

Exemple :

```
program DEMO_IMP;   (* démonstration emploi imprimante *)
Uses Crt, Printer;
Var Nom : string[30];
begin
     Write('Votre nom : ');
     ReadLn(Nom);
```

```
    WriteLn(Lst,'Voici votre nom imprimé : ');
    WriteLn(Lst,'Monsieur, Madame ou Mademoiselle :',Nom);
    Write(Lst,'Fin de programme.');
end.
```

Comme vous pouvez en juger, ce programme ne présente aucune difficulté. Ah, oui ? Le type *string ?* C'est pour introduire une "chaîne" de caractères, un mot ou une phrase. Donc utiliser des caractères et non des nombres. Le comportement de *string* est très proche de celui de *Array*, comme vous vous en doutez. Il y a de nombreuses fonctions et instructions pour traiter les chaînes. Nous n'en parlerons pas dans ce livre (mis à part les instructions graphiques des chapitres suivants) : c'est plus une question de pratique qu'une difficulté réelle.

Passons à l'emploi du lecteur de disquettes. C'est du même genre : il faut définir le "canal" sur lequel on va travailler. Mais il y a une difficulté supplémentaire : la liaison se fait indirectement, par le biais d'un fichier en mémoire. Il est donc nécessaire d'annoncer d'abord comment se fait cette liaison.

Disons qu'il y a un "fichier-disquette" et un "fichier-mémoire". La laison entre les deux est faite grâce à l'instruction ***Assign*** :

Assign(nom du fichier-mémoire, nom du fichier-disquette);

Le "fichier-disquette" prend un nom qui suit les mêmes règles que tous vos autres fichiers : pas plus de huit signes, un point, une extension de trois signes. On peut le faire précéder de l'indication du lecteur utilisé. Exemple : A:VIE.PAS. Quant au "fichier-mémoire", vous le dénommez comme vous l'entendez.

Cette indication étant donnée au programme, il faut préciser ce que l'on veut faire : créer, c'est-à-dire aussi bien ré-écrire, un fichier, ou simplement le ressortir : *ReWrite* dans le premier cas, *Reset* dans le second.

On peut alors travailler sur le "fichier-mémoire", écrire dedans par ***Write(nom du fichier-mémoire,*** *ce qu'on a à écrire);* etc. Vous voyez bien la ressemblance avec l'emploi de l'imprimante.

Une fois qu'on a fini de travailler avec ce fichier, on le signifie poliment à l'ordinateur par un ***Close(nom du fichier-mémoire)*** qui lui indique qu'il faut "fermer" le fichier.

Tout est dit, ou presque. Passons à un exemple qui nous est utile : la création d'un fichier pour conserver nos plus belles populations du "jeu de la vie".

Ce fichier sur disque, nous le dénommerons VIE.TAB (un tableau pour la vie...). Nous nommerons *Conserve* le "fichier-mémoire" que nous créons, mais vous pouvez mettre *MiseEnBoite* ou *LesMemoiresDeMesVies* si cela vous

chante. Il faut bien sûr le déclarer. Ce qui demande un peu de réflexion : le plus simple, est de prendre un fichier (avec le mot réservé *File of*) de tableaux copiés sur *Organe*, ce qui nous permet de copier directement *Memoire* sans se casser la tête. Défaut : on prend 1920 octets à chaque fois, alors que bien souvent on n'a qu'une trentaine de points, au plus, dont il faut conserver l'emplacement! C'est vraiment du luxe... On reparlera brièvement de cette question en fin de chapitre.

En fonction de ce que nous avons dit plus haut, notre programme (isolé) de création d'un fichier pour le "jeu de la vie" s'écrit :

```
program CRFICHVI;    (* création fichier jeu de la vie *)
Uses Crt;
type Organe = Array[1..1920] of boolean;
type Fichier = File of Organe;
Var Conserve : Fichier;
begin
     Assign(Conserve,'VIE.TAB');
     Rewrite(Conserve);
     Close(Conserve);
end.
```

Allez-y ! Tapez ce mini-programme, et faîtes-le fonctionner : il va vous créer le fichier VIE.TAB dont nous aurons bientôt besoin. Vérifiez que le résultat est bon : il suffit de faire, dans le sous-menu File de Turbo Pascal, une demande de "Directory" avec le masque *.TAB et le catalogue qui s'affiche vous donne bien : VIE.TAB...

Maintenant que le fichier-disque est créé, encore faut-il l'utiliser. Et d'abord le remplir. Nous allons ajouter une procédure spéciale pour cela dans le programme du jeu de la vie. Attendez quand même pour le faire : le listing complet de "LAVIE" est en fin de chapitre. Cette procédure est brève :

```
procedure Enregistre;      (* enregistrement population "intéressante" *)
begin
     (* AMELIORATION SOUHAITABLE : CREATION AUTOMATIQUE *)
     Assign(Conserve,'VIE.TAB'); Reset(Conserve);
     Numero:=FileSize(Conserve); Seek(Conserve,Numero);
     Write(Conserve,Memoire);Close(Conserve);
     Write('Numéro d''enregistrement : ',Numero);
     Delay(3000);
end;
```

Les deux premières lignes ne vous posent aucun problème, espérons-le. La troisième est là pour qu'on se souvienne, un de ces jours, que le mieux serait

d'intégrer le programme CRFICHVI dans LAVIE. Pas tout à la fois s'il vous plaît ! La quatrième est là pour spécifier la liaison fichier-disque = > fichier-mémoire et qu'on ouvre le fichier sans le détruire comme on le ferait si on mettait *Rewrite.*

La cinquième ligne nous fait découvrir deux instructions nouvelles : *File-Size(nom du fichier)* nous donne la taille, le nombre d'enregistrements, de notre fichier. Indispensable : quand on ouvre un fichier, on se place en début de fichier. Si on n'y prend pas garde, on écrira toujours sur le premier enregistrement, ce qui n'est pas très efficace pour accumuler les données... Une fois connue la taille du fichier, il suffit d'aller se mettre à la fin, de se déplacer grâce à *Seek.* Remarquez qu'on aurait pû le faire directement, par *Seek(FileSize(Conserve));* mais l'utilisateur n'aurait pas pû avoir connaissance du nombre de populations mises "en boîte".

La sixième ligne est évidente : on écrit dans *Conserve* ce qui est dans *Memoire,* et on referme le fichier. Précisons quand même que rien n'oblige à refermer le fichier de suite, d'autant plus que MSDOS supporte très bien d'en avoir plusieurs ouverts simultanément...

La fin de la procédure ne pose pas plus de problème : affichage à l'écran du numéro de la population sur la disquette, avec le temps de lire...

La procédure *Relecture* (devinez à quoi elle sert) n'est pas plus compliquée. Bien sûr, on met *Read* à la place de *Write* pour lire au lieu d'écrire. Vous remarquerez que, là encore, l'instruction se comporte en n'utilisant que la console (le clavier pour lire, l'écran pour écrire) s'il n'y a pas d'indication de canal à utiliser. Seule petite difficulté : la numérotation dans un fichier commence à zéro, ce qui nous oblige à un petit exercice de réflexion pour bien numéroter quand *Seek* nous envoie à un numéro donné, c'est-à-dire celui d'après celui qu'on voudrait...

```
procedure Relecture;
begin
     (* AMELIORATION A APPORTER : VERIFICATION VALIDITE ! *)
     Assign(Conserve,'VIE.TAB'); Reset(Conserve);
     ClrScr; GotoXY(10,1);
     Write(FileSize(Conserve),' populations  déja enregistrées.');
     GotoXY(10,3); Write('Numéro de la population demandée : ');
     Read(Numero);
     Seek(Conserve,Numero-1); Read(Conserve,Memoire); Close(Conserve);
end;
```

Vous pouvez maintenant prendre le listing "final" qui se trouve à la fin de ce chapitre. "LAVIE" est fabriquée à partir de "VIE", et nous avons mis un "menu" pour que l'utilisateur ait la possibilité de reprendre une population

déjà existante ou en créer une lui-même. Nous utilisons un *Case ... of*, cette fois-ci avec des constantes pour les différents cas. Chacun des cas donne lui-même un véritable petit sous-programme (qui peut être vide, voyez après *else*).

Petit défaut : si on presse 'Q' par inadvertance, on quitte. Petite difficulté : pas de point-virgule avant *else* : le motif consiste, en quelque sorte, à prévenir le compilateur que ce n'est pas fini... Notez quand même que *else* dans *case* est une extension de Turbo Pascal par rapport au Pascal standard, si tant est qu'il en existe un ; certains langages Pascal emploient *otherwise*, d'autres n'ont pas du tout de possibilité d'utiliser un "autre cas" global (dur, dur !). On disait ça pour votre culture personnelle.

Parlons un peu des améliorations possibles, voire souhaitables.

Sur le plan de la structure du programme, d'abord. Vous pouvez ajouter, avec un deuxième menu, des options d'affichage : commandé par pression d'une touche au lieu d'être automatique, en décoration (c'est-à-dire sans rien d'autre à l'écran que les thêtas), et en automatique comme nous l'avons fait. Prévoir la possibilité de ne travailler que sur un nombre donné de populations. Tout cela a été fait dans "Turbo Pascal pas à pas" pour la version 3, mais nous ne le ferons pas ici, faute de place : il faut bien parler un peu des graphismes !

Sur le plan de la sécurité de fonctionnement ensuite. Il est navrant de voir un programme se "planter" parce que le programmeur n'a pas prévu que l'utilisateur pourrait avoir des idées farfelues, du genre introduire un "è" quand on lui demande un numéro de population, ou utiliser ce jeu sans avoir de fichier de populations sur sa disquette. Turbo Pascal nous permet de remédier à ce genre de problème. Il suffit d'utiliser la vérification des entrées et sorties. On désactive le contrôle automatique par : ***{$I-}*** tout seul sur sa ligne, on effectue l'opération délicate (par exemple lecture du numéro de population), on réactive le contrôle par ***{$I+}*** puis on teste le résultat par *IOResult* (Input=entrée, Output=sortie). Si le résultat est différent de zéro, c'est qu'il y a problème, donc qu'il faut suivre un cheminement particulier.

Exemple pour notre programme :

```
Write(FileSize(Conserve),' populations  déja enregistrées.');
GotoXY(10,3); Write('Numéro de la population demandée : ');
{$I-}
Read(Numero);
{$I+}
if (IOResult <> 0) then begin
   Write('Demande impossible à satisfaire');
   Close(Conserve);
   Delay(2000);
   Relecture;
   end;
```

On peut en faire des choses ! Nous vous conseillons de reprendre votre programme en le truffant de *{$I-}* et de *{$I+}*, éventuellement en interprétant finement ce que retourne IOResult, sans se contenter de la seule valeur non-zéro.

Plus délicate est l'amélioration possible du fichier des populations. Il serait bon tout à la fois de faire plus petit et de mettre des indications supplémentaires du genre date, nom de l'auteur, etc., toutes indications qui sont elle-mêmes de type bien différent. Pas de mystère : il faut utiliser le type *record*. N'étant pas angliciste professionnel, nous avons regardé de près un dictionnaire, et constaté avec joie que "record" signifie le plus souvent "dossier", et que si c'est "enregistrement" que l'on prend, c'est dans le sens "enregistrer une plainte", par exemple. Le sens "enregistrement sur un support magnétique (ou disque)" est récent, mais c'est le plus usité actuellement en français pour "enregistrement". Hélas ! Car c'est bien d'un dossier qu'il s'agit, avec ses sous-dossiers dont les contenus peuvent être très différents. L'utilisateur définit lui-même son dossier, son *record*, un peu comme on l'a fait pour *Array*.

Exemple :

```
Population = record
             Auteur : string[30];
             Date : Array[1..3] of byte;
             TableauPlaces : Array[1..100] of byte;
             end;
```

Cette déclaration permet de réduire la place prise en mémoire : un enregistrement ne dépasse que de peu 130 d'octets (au lieu de 1920 !), tout en donnant des indications sur l'auteur et la date. Pour utiliser cela, il vous faut modifier le programme en conséquence, pour introduire la saisie du nom, de la date (ou la prendre sur l'horloge interne), et répertorier les cellules mises en place (100 cellules, c'est déjà pas mal).

Pour ressortir et utiliser un *record* , il faut prendre l'instruction ***with***, par exemple pour ci-dessus : *with Population do* etc.

Il va sans dire que ce type vous sera fort utile si vous voulez perfectionner le jeu de réflexes, en stockant les résultats sur disquette. Et aussi pour tous les programmes que vous ferez avec des fiches (sur une personne, un salaire, un événement, etc.). Bon courage, bon plaisir !

Nous avons vu :

- comment utiliser l'imprimante ;
- comment utiliser le lecteur de disquettes ;
- les instructions *Assign, Reset, Rewrite, Close, Seek, FileSize.*

Nous avons aperçu :

- le contrôle des entrées/sorties ;
- le type *string* et le type *enregistrement.*

```
program LAVIE;                (*       jeu de la vie
                              pour "Turbo Pascal V 4.0 pas à pas"
                                     P.S.I.   1988          *)

Uses Crt;

var CellX, CellY : byte;                          (* coordonnées cellule *)
    Compteur : word;                                (* "âge" population *)

type Organe = Array[1..1920] of boolean;        (* type d'une population *)
var  Bestiole, Memoire, Change : Organe;          (* tableaux de travail *)
var Reponse : char;                             (* réponse aux questions *)

type Fichier = File of Organe;                    (* pour gestion disque *)
var Conserve : Fichier;
    Numero : integer;

(*-------------------------------------------------------------------------*)
```

```
procedure Nettoyage;                    (* nettoyage préalable des tableaux *)
var i : word;
begin
     for i := 1 to 1920 do                    (* 1920 = 24 fois 80 ... *)
        begin
             Bestiole[i] := False;          (* remise à zéro population  *)
             Memoire[i] := False;           (*    "       " mémorisation *)
             Change[i] := False;            (*    "       " utilitaire   *)
        end;
end;

(*-------------------------------------------------------------------------*)
procedure Saisie;                        (* saisie des positions initiales *)
var Reponse : char;
begin
     Reponse := UpCase(ReadKey);             (* trois possibilités utiles *)

     if Reponse = ' ' then begin           (* validation d'un emplacement *)
        Write('X');                                      (* affichage *)
        GotoXY(CellX,CellY);                          (* retour curseur *)
        Memoire[80*(CellY-1)+CellX] := True;          (* mise en mémoire *)
     end;

     if Reponse = #0 then begin (* touche "caractère étendu" *)
        Reponse := ReadKey;       (* analyse du deuxième code *)
        Case Reponse of
             #72 : CellY := CellY - 1;              (* haut    *)
             #80 : CellY := CellY + 1;              (* bas     *)
             #75 : CellX := CellX - 1;              (* gauche  *)
             #77 : CellX := CellX + 1;              (* droite  *)
             #83 : begin                            (* annule  *)
                        Memoire[CellX + 80*CellY] := False;
                        Write(' ');
                   end;
             else Saisie;                  (* mauvaise touche... *)
         end;
         GotoXY(CellX,CellY);                     (* déplacement *)
     end;

     if Reponse<>'F' then Saisie;    (* on continue la saisie *)
end;

(*-------------------------------------------------------------------------*)
```

```
procedure MiseEnPlace;                                    (* écran de saisie *)
begin
     ClrScr;
     GotoXY(12,25);
     HighVideo;
     Write('Validation : <espace>     Effacement : <Del>      Fin : F');
     NormVideo;
     CellX := 40; CellY := 12;                              (* centre écran *)
     GotoXY(CellX,CellY);
     Saisie;
end;

(*-------------------------------------------------------------------------*)
procedure Initialise;          (* initialisation tableau  bestiole ... *)
var i : word;
begin
     Compteur := 1;                        (* ... et "âge" population. *)
     GotoXY(12,25);
     Write('            Pour finir : presser une touche.              ');
     for i:=1 to 1920 do Bestiole[i]:=Memoire[i];
end;
(*-------------------------------------------------------------------------*)
procedure EtatCivil;                     (* fait évoluer une population *)
var i : word;                           (* indice pour la boucle *)
K : byte;                         (* caractérise environs cellule *)
begin
  GotoXY(1,25); Write(Compteur); Inc(Compteur);
  for i:=81 to 1839 do begin   (* on évite première et dernière ligne *)
    K:=Ord(Bestiole[i])+Ord(Bestiole[i-80])+Ord(Bestiole[i+80]);
    K:=K+Ord(Bestiole[i-1])+Ord(Bestiole[i-81])+Ord(Bestiole[i+79]);
    K:=K+Ord(Bestiole[i+1])+Ord(bestiole[i-79])+Ord(Bestiole[i+81]);
    Change[i]:=NOT(K=4) AND (Bestiole[i] XOR (K=3));
  end;

  for i:=1 to 1920 do      (* mise à jour tableaux  -  affichage *)
    if Change[i] then begin
      Bestiole[i] := NOT Bestiole[i];
      Change[i] := False;        (* au passage : remise à False *)
      CellY := 1 + Trunc(i/80);
      CellX := i - 80*(CellY-1);
      GotoXY(CellX,CellY);
      if Bestiole[i] then Write(#233) else Write(' ');
  end;
end;
(*-------------------------------------------------------------------------*)
```

```
procedure Enregistre;      (* enregistrement population "intéressante" *)
begin
     (* AMELIORATION SOUHAITABLE : CREATION AUTOMATIQUE *)
     Assign(Conserve,'VIE.TAB'); Reset(Conserve);
     Numero:=FileSize(Conserve); Seek(Conserve,Numero);
     Write(Conserve,Memoire);Close(Conserve);
     Write('Numéro d''enregistrement : ',Numero);
     Delay(3000);
end;
(*-------------------------------------------------------------------------*)
procedure Relecture;
begin
     (* AMELIORATION A APPORTER : VERIFICATION VALIDITE ! *)
     Assign(Conserve,'VIE.TAB'); Reset(Conserve);
     ClrScr; GotoXY(10,1);
     Write(FileSize(Conserve),' populations  déja enregistrées.');
     GotoXY(10,3); Write('Numéro de la population demandée : ');
     Read(Numero);
     Seek(Conserve,Numero-1); Read(Conserve,Memoire); Close(Conserve);
end;
(*-------------------------------------------------------------------------*)
procedure Final;
begin
     GotoXY(10,1);
     WriteLn('Conservation de la population initiale ? (O/N)');
     Reponse := UpCase(ReadKey);
     if Reponse = 'O' then Enregistre;
end;
(*-------------------------------------------------------------------------*)
procedure Introit;
var i, X, Y : word;
begin
     ClrScr;
     GotoXY(20,10); Write('JEU DE LA VIE');
     GotoXY(15,14); Write('Création d''une population   : C');
     GotoXY(15,16); Write('Population déja enregistrée : D');
     GotoXY(15,20); Write('Pour quitter : Q');
     Reponse := UpCase(ReadKey);
     Case Reponse of
          'C' : begin
                     Nettoyage;
                     MiseEnPlace;
                     Initialise;
                     while not KeyPressed do EtatCivil;
                     Reponse := UpCase(ReadKey);
                     Final;
                end;
```

```
          'D' : begin
                    Nettoyage;
                    Relecture;
                    ClrScr;
                    for i:=1 to 1919 do
                       if Memoire[i] then Write('X') else Write(' ');
                    Initialise;
                    while not KeyPressed do EtatCivil;
                    Reponse := UpCase(ReadKey);
                end
           else ;
      end;
end;

(*═══════════════════════════════════════════════════════════════════════*)
begin
     Reponse := 'z';
     while (Reponse<>'Q') do Introit;
end.
(*FFFF
```

CHAPITRE 10

PASSONS AU GRAPHISME

Prévenons de suite : les explications vont être rapides. On n'est plus au chapitre 1 ! Il vaut mieux que vous changiez de disquette. Cette fois, il vous faut tous les fichiers ayant l'extension .BGI. Du moins si vous désirez que ce programme tourne sur toutes les visus.

Nous vous l'avons dit, la version 4 de Turbo Pascal est prévue pour fonctionner avec la plupart, pour ne pas dire la quasi-totalité, des compatibles PC. Or ces derniers sont très différents sur le plan de l'image. On va du PC de base, avec de toutes petites cellules, au super-developpé avec une myriade de couleurs et des pixels partout, avec tout ce qu'il faut pour les PS présents et à venir. Et la gestion de tout ça n'est pas simple. Leurs fonctionnements sont tout à fait différents, autant que leurs affichages.

Ne rentrons pas dans les détails, même si certains se montrent parfois importants :

- affichage CGA, avec versions monochrome ou couleur, cette dernière vous sortant 320 x 200 points en quatre couleurs ou 640 x 200 en monochrome

(ce qui veut dire, comme chacun sait, en deux couleurs, celle des points et celle du fond, sinon on ne verrait rien) ;

- affichage EGA, avec sa mémoire auxiliaire et ses seize couleurs avec une définition de 640 sur 350 pixels (les plus petits "points" possibles à l'affichage). Pas mal ;

- le mode Hercule, monochrome, définition 720 x 350. C'est assez net ;

- le mode VGA, qui n'est pas bien vieux, et qui travaille avec 640 x 480 points. On se rapproche des performances d'une mauvaise TV... Il n'y a pas encore de quoi s'extasier !

Bien sûr, on pourrait faire un programme pour chaque mode de fonctionnement. Mais il vaut quand même mieux se débrouiller pour que le programme que vous écrivez puisse passer sur n'importe quelle "visu", même si ce n'est pas la vôtre.

Vous pouvez reprendre le programme ARTY4.PAS de vos disquettes Borland, celui que nous avons utilisé au chapitre 1. Tout son début est prévu pour lui permettre de fonctionner avec presque n'importe quel standard graphique. Vous remarquerez au passage que les programmeurs n'ont pas hésité à mettre des noms explicites, du moins pour les anglophones.

Le programme que nous étudierons s'efforcera de travailler sur tous ces standards.

Au juste, quel programme ?

On va essayer de faire joli sans pour autant tomber dans du trop compliqué. Un peu d'aléatoire, des jolis dessins, de la couleur (pour les visus qui l'ont). Nous avons choisi les courbes de Lissajous... Vous connaissez ? Même si vous n'avez que peu de connaissances en mathématique ou en physique, il ne vous est pas interdit de savoir que Lissajous est né à Versailles en 1822, est mort en 1880, et qu'il étudia la composition des mouvements vibratoires en optique. C'est pourquoi les courbes qui portent son nom son très connues des électroniciens (non pas à cause de l'optique, mais des compositions des vibrations). En bref, une courbe de Lissajous est donnée par les coordonnées du point courant avec u comme paramètre :

$$x = a \cos(cu) \qquad y = b \sin(du)$$

Suivant les valeurs relatives de a, b, c et d, on obtient des tas de jolies choses. Mais on n'est pas là pour faire des maths !

Notre propos pourrait être assez simple : tracer une courbe de Lissajous en "entrant" les valeurs de a, b, c et d.

On va quand même faire mieux que cela. Se donner des tailles variables pour

dessiner ces courbes. Les inclure à chaque fois dans un joli rectangle. Faire varier les couleurs. Et, au chapitre suivant, mettre un texte publicitaire avec des dimensions et des caractères variables (gothique, par exemple...).

Pour l'instant, établissons notre programme. Il nous faudra :

- une première procédure pour initialiser l'écran. On l'utilisera à chaque affichage d'une nouvelle "ellipse", pour qu'un écran couleur CGA puisse changer de palette (en mode CGA, on joue sur quatre palettes de quatre couleurs) ;
- une deuxième pour le calcul des "dimensions" de l'écran et ses conséquences sur ses sous-multiples : pour avoir un résultat correct à l'écran, il faut que le rectangle que l'on trace soit au moins égal à la moitié de l'écran, et au plus à l'écran. Et aussi pour fixer la valeur des paramètres a, b, c et d que nous avons définis ci-dessus. Comme nous ne jouons pas au mathématicien acharné, on donnera des noms relativement explicites. On profitera de cette procédure pour fixer la couleur du tracé. Tout cela en utilisant la fonction *Random* ;
- une troisième pour effectuer les tracés, celui du rectangle et celui de la courbe.

Il n'est pas nécessaire dans un livre comme celui-ci de donner le détail des choix qui ont été fait dans le programme ci-après. Disons par exemple que prendre 2000 points est suffisant pour avoir un tracé précis de la courbe dans la plupart des cas, sans que l'affichage dure trop longtemps. Même chose pour les paramètres de sinus et cosinus : les 23500 combinaisons possibles semblent en nombre suffisant.

NOTION DE FONCTION

Parlons plutôt de la notion de fonction, que nous utilisons ici bien que son emploi ne soit pas vraiment indispensable.

Une fonction ressemble à une procédure. Mais on doit préciser ce qu'elle reçoit et ce qu'elle renvoie. On écrit :

function Nom_Fonction(variable_reçue : type) : type du renvoi;

Exemple simple :

```
function triple(mult : real) : real;
begin
    triple := 3*mult;
end;
```

Vous pouvez employer cette fonction dans un programme, toutes les fois où vous voudrez multiplier un réel par trois. Il vous suffira d'écrire : ***MaVariable := Triple(SaVariable);*** pour que ce produit se fasse. Bon, d'accord, ce n'est pas un avantage énorme. Mais, quand un calcul est tellement complexe qu'on en perd le fil, il n'est pas mauvais de lui donner un nom explicite ! Et, surtout, vous avez remarqué qu'une fonction n'a pas besoin de déclaration extérieure de variable. C'est-à-dire que si vous fabriquez une fonction particulièrement "tordue", elle peut vous servir dans n'importe quel programme, comme ça, sans modification. Intéressant, non ? Vous pouvez ainsi la mettre sans peine dans une Unité qui vous est particulière. Du pain sur la planche...

Voici le programme complet, que nous avons nommé LISS1 parce qu'il y a un LISS2.

```
program LISS1;          (* en hommage à Lissajous.
             Ce programme trace des "ellipses" encadrées par
             des rectangles en différentes couleurs et tailles. *)
Uses Crt,
     Graph;

var ParamCos, ParamSin : real;          (* définies par auteur *)
    Couleur, MaxCouleur,                (*      idem           *)
    AmplitudeX, AmplitudeY,
    DemiLargeur, DemiHauteur,
    QuartLargeur, QuartHauteur : word;
    GraphDriver, GraphMode : integer;  (* noms copiés sur Borland *)

(*-------------------------------------------------------------------*)
procedure InitialiseEcran;

(* trouve quel est le gestionnaire d'écran en place et
   prend le mode couleurS en CGA. Tous "drivers" utilisables. *)

begin
     DetectGraph(GraphDriver,GraphMode);
     Case GraphDriver of
          CGA : GraphMode := Random(4);   (* refuse donc le mode 4 *)
          MCGA : GraphMode := Random(4);  (*          idem         *)
          EGA : GraphMode := 1;
          VGA : GraphMode := 2  (* rappel : pas de ';' avant else *)
          else GraphMode := 0;
          end;
     InitGraph(GraphDriver,GraphMode,'');
end;

(*-------------------------------------------------------------------*)
```

```
procedure Parametres;   (* calcul des paramètres de "l'ellipse" *)
begin

     (* paramètres de l'écran, imposés par le driver et le mode *)
     MaxCouleur := GetMaxColor;
     DemiLargeur := GetMaxX div 2;     (* de l'écran, en pixels *)
     DemiHauteur := GetMaxY div 2;
     QuartLargeur := DemiLargeur div 2;
     QuartHauteur := DemiHauteur div 2;

     (* paramètres aléatoires : *)

     (* paramètres pour la courbe *)
     ParamCos := 1 + (Random(30)/(1 + Random(5)) ) ;
     ParamSin := 1 + (Random(30)/(1 + Random(5)) ) ;
     Couleur := 1 + Random(MaxCouleur);

     (* paramètres généraux *)
     AmplitudeX := QuartLargeur + Random(QuartLargeur);
     AmplitudeY := QuartHauteur + Random(QuartHauteur);
end;

(*-------------------------------------------------------------------*)

function CalcX(z : integer) : word;
(* cette fonction retourne la valeur de X *)
begin
     CalcX := DemiLargeur + Trunc(AmplitudeX * cos(z*ParamCos));
end;
(*-------------------------------------------------------------------*)
function CalcY(z : integer) : word;
(* celle-ci la valeur de Y ... *)
begin
     CalcY := DemiHauteur + Trunc(AmplitudeY * sin(z*ParamSin));
end;

(*-------------------------------------------------------------------*)
```

```
procedure Trace;
(* trace le rectangle et la courbe de Lissajous qui
   correspondent aux valeurs calculées.              *)

(* variables qu'il n'est pas utile de déclarer globalement *)
var RectHGX, RectHGY, RectBDX, RectBDY : word;
    i : integer;

begin
     (* tracé du rectangle *)
     RectHGX := DemiLargeur - AmplitudeX - 1;
     RectHGY := DemiHauteur - AmplitudeY - 1;
     RectBDX := DemiLargeur + AmplitudeX + 1;
     RectBDY := DemiHauteur + AmplitudeY + 1;

     SetColor(1 + Random(MaxCouleur));   (* couleur tracé rectangle *)
     Rectangle(RectHGX,RectHGY,RectBDX,RectBDY);

     (* tracé de la courbe de Lissajous *)
     for i:=1 to 2000 do PutPixel(CalcX(i),CalcY(i),Couleur);

end;

(*=============================================================*)

begin
     Randomize;
     while not KeyPressed do
          begin
               InitialiseEcran;
               Parametres;
               Trace;
          end;
     CloseGraph;
     RestoreCrtMode;
end.

(*FFFF
```

naire d'écran ("driver") et mode sont à utiliser. Elle fonctionne de pair avec *InitGraph(GraphDriver,GraphMode,'chemin');*. Ce n'est pas très simple mais indispensable à bien comprendre si on veut réellement faire du graphisme avec Turbo Pascal. Expliquons un peu.

Votre gestionnaire d'écran est d'un des modèles que nous avons rapidement définis. On peut appeler *InitGraph* sans passer par *DetectGraph*, auquel cas *InitGraph* le fera sans vous demander votre avis, et prendra automatiquement ce qui donne la meilleure définition de l'écran. L'os, car os il y a, est qu'en CGA on prend certes la meilleure définition mais on perd en même temps la couleur, ce qui est bien dommage ! Si on fait appel à *DetectGraph*, on connait alors les valeurs qui correspondent au gestionnaire de fichier, et décider de ce que l'on va faire. L'appel qui suit *InitGraph* ne provoquera pas un appel à *DetectGraph*, considérant que c'est déjà fait.

Merci Borland, les valeurs de *GraphDriver* sont pré-définies. C'est-à-dire que l'on peut écrire clairement ce que l'on veut faire. Il n'y a que dans les deux cas CGA et MCGA qu'il y a besoin de jouer sur *GraphMode*, pour pouvoir changer de palette de couleurs. Il y a quatre palettes possibles, numérotées de 0 à 3, d'où le *Random(4)*. Précisons bien : si on ne mettait pas *GraphDetect*, *InitGraph* donnerait de suite en CGA la valeur maxi (4) et on se trouverait en haute définition monochrome. En EGA ou VGA, pas de problème : il y a seize couleurs en mode haute définition. En Hercules non plus : c'est toujours en monochrome...

Le reste de cette procédure ne pose pas de problème. ***InitGraph*** prend le bon gestionnaire d'écran (c'est pour cela qu'il vous faut tous les .BGI), donne au mode graphique la bonne valeur. L'espace entre apostrophes à la fin ? On pourrait y mettre le chemin, le "path", nécessaire à la recherche du gestionnaire d'écran (inutile pour nous : tout est sur la même disquette). Tout est vraiment prévu !

La suite du programme est simple. Nous avons essayé de mettre des noms assez explicites, pour vous faciliter la lecture.

Tout est bien facile désormais. Sauf peut-être la fonction ***GetMaxColor*** ? Elle nous renvoie gentiment la plus forte valeur possible pour les couleurs (de 1 à 3 en CGA, de 1 à 16 en EGA ou VGA). N'oubliez pas que 0 = noir... D'où le "1+Random..." dans les calculs de couleurs, pour éviter ce noir.

Les fonctions ***GetMaxX*** et ***GetMaxY*** renvoient les valeurs maximales que peuvent prendre les coordonnées d'un pixel en haute définition (par exemple 639 et 349 en EGA). Quant au ***div 2*** c'est pour avoir un résultat entier, alors que si vous mettez */2* vous aurez en retour un nombre réel que vous serez obligé de triturer avec *Trunc* pour qu'il soit acceptable par nos fonctions graphiques.

L'emploi des procédures et fonctions de Turbo Pascal version 4 que sont *Rectangle(CoinEnHautGauche,..,..,..)* avec couleur donnée par ***SetColor*** ne mérite pas d'explication, pas plus que celui de *PutPixel(horiz,vertic,couleur).*

Tout en fin de programme, ***CloseGraph*** et ***RestoreCrtMode*** nous font revenir à l'état initial de l'écran (fin du graphique, retour au mode courant).

Vous pouvez essayer de ramener ce programme à vingt lignes, ou moins. C'est même un bon exercice intellectuel. Que celui qui y arrive en moins d'une heure écrive à l'auteur dont les nerfs fragiles n'ont pas résisté plus de trente minutes dans cette recherche... qui conduit à un listing à peu près inextricable !

Des améliorations sont possibles, bien sûr. Mettre un bandeau pour que l'utilisateur sache qu'il faut presser une touche pour arrêter l'affichage. Pouvoir entrer des valeurs différentes de 30 et 5 pour les calculs de ***ParamCos*** et ***ParamSin***. Il serait d'ailleurs bon de définir ces deux valeurs comme constantes dans une déclaration : ce serait plus lisible. On mettrait :

```
const : Numerateur = 30,
        Denominateur = 5;
```

et, dans la procédure :

ParamCos := 1 + (Random(Numerateur) / (1 + Random(Denominateur));
ParamSin := etc. (les deux ne sont pas égales : *Random* ne donne pas le même résultat !).

Vous pouvez (si ce n'est déja fait) prendre votre disquette, la troisième, et fabriquer ce programme. Dans un premier temps, il n'est pas nécessaire de taper toutes les remarques : vous avez ce livre pour le listing détaillé.

Nous avons vu :

- l'utilisation des gestionnaires d'écran ;
- les instructions graphiques de l'Unité *Graph :*
  . *DetectGraph(GraphDriver,GraphMode) ;*
  . *InitGraph(GraphDriver,GraphMode,'path') ;*
  . *GetMaxX, GetMaxY, GetMaxColor ;*
  . *CloseGraph, RestoreCrtMode.*

CHAPITRE 11

FRANCHISSONS LE MUR DES 64 KO

Comme vous le savez, un programme fabriqué par la version 3.0 de Turbo Pascal ne pouvait dépasser 64 Ko qu'au prix d'acrobaties pas très évidentes... Avec la version 4, cette fantaisie est assez facile à réaliser.

De plus, il est intéressant d'avoir une première approche de la manière dont on peut constituer ses propres "Units", avec des procédures personnelles toutes faites. Nous ne donnerons certes pas ici tout le détail de ce qu'il faut faire : nous nous contenterons de donner un exemple de fabrication d'unités sur un exemple concret. Pour vous donner l'eau à la bouche...

Notre propos est simple : il s'agit d'ajouter une inscription dans le rectangle tracé par LISS1. Pas n'importe comment, quand même. Pour bien faire, il faudrait que cette inscription soit de taille aléatoire, et écrite dans tous les caractères que Borland met à notre disposition : Gothique, Sans Serif, Triplex (triple dessin), Petits (qu'on fera agrandir...) et bien sûr les caractères standard. Toutes ces fontes (c'est le terme même employé par les impri-

meurs) sont données sur la disquette Borland Numéro 3, avec l'extension ".CHR".

Il faut donc que ces fontes soient sur votre disquette. Si ce n'est pas le cas, mettez-les y vite. Et, puisque vous êtes sous MS-DOS, profitez-en pour mettre aussi tous les fichiers en .BGI s'ils n'y sont pas déjà. Prenons de l'avance : mettez aussi les fichiers BINOBJ.EXE, DRIVERS.PAS et FONTS.PAS. Comme sur votre disquette vous avez déjà GRAPH.TPU, TURBO.EXE, TURBO.TPL, et aussi LISS1.PAS, elle commence à donner des signes d'embonpoint. On s'en sortira, sauf si vous avez créé un LISS1.EXE qui risque de prendre trop de place.

En pratique, il faut et il suffit que vous ayez sur cette disquette :

TURBO.EXE
TURBO.TPL
BINOBJ.EXE
GRAPH.TPU
LISS1.PAS
FONTS.PAS
DRIVERS.PAS
ATT.BGI
CGA.BGI
EGAVGA.BGI
HERC.BGI
PC3270.BGI
GOTH.CHR
LITT.CHR
SANS.CHR
TRIP.CHR

Pour pouvoir écrire à l'écran, il nous faut ajouter une procédure. Elle est mise en fin de programme, qui devient :

```
procedure Ecriture;
begin
     SetTextJustify(CenterText,CenterText);
     SetTextStyle(Random(5),HorizDir,1+Random(10));
     SetColor(1 + Random(MaxCouleur));
     OutTextXY(DemiLargeur, DemiHauteur, 'P.S.I.');
end;

(*=============================================================*)
```

```
begin
     Randomize;
     while not KeyPressed do
          begin
               InitialiseEcran;
               Parametres;
               Ecriture;
               Trace;
          end;
     CloseGraph;

     RestoreCrtMode;
end.

(*FFFF
```

Constat : à chaque fois que le programme change de fonte, il y a appel au lecteur de disquettes. Ce qui ne fait pas très sérieux, surtout si vous avez l'intention de laisser tourner ce programme pendant des heures... Vous pouvez essayer de compiler sur disque, rien n'y fait.

Il faut à ce défaut porter remède. C'est-à-dire intégrer les fontes (et, au passage, les gestionnaires d'écran) dans le programme pour qu'il devienne totalement indépendant (et transportable).

Ce qui est moins facile que ci-dessus. Disons même que ce qui suit est là pour vous donner envie d'aller encore plus loin avec Turbo Pascal et non pas pour vous donner dans le détail tout ce qu'il faudrait savoir. Nous allons utiliser, avec le minimum d'explications, des programmes déjà installés dans vos disquettes Borland.

Votre disquette est prête ? Allons-y...

Il nous faut tout d'abord créer des fichiers avec l'extension **.OBJ.** C'est-à- dire des fichiers "objets" qui peuvent être reliés à un programme source (extension .PAS) pendant sa compilation. On en fera ensuite des fichiers ***.TPU** qui nous servirons de "Unit" qu'on pourra appeler avec "Uses". C'est-à-dire que, ce que nous allons faire ici vous resservira sûrement !

Faisons-le pour les fontes. Nous allons suivre ici rigoureusement les dénominations employées par Borland dans ses exemples **FONTS.PAS** et **GRLINK.PAS**, quoique notre méthode soit plus simple que celle de GRLINK qui emploie sans prévenir le programme TPC.EXE, ce qui n'a rien d'obligatoire.

Créons des fichiers objets à partir des *.CHR. Pour ce faire, nous employons le programme ***BINOBJ.EXE*** qu'il faut donc avoir sur la disquette. Repassez sous DOS, par OS Shell du menu File. Vous entrez, en respectant bien les espaces et en une seule fois :

BINOBJ GOTH.CHR GOTH GothicFontProc

Mais oui ! Détaillons.

BINOBJ est l'appel au programme du même nom, qui a besoin d'autres indications.

GOTH.CHR est la première de ces indications : c'est le nom du programme dont on veut tirer un programme OBJ.

GOTH est la deuxième : c'est le nom du programme qui sera suivi de OBJ. Si on avait mis GOT et non GOTH, ce programme deviendrait GOT.OBJ et non GOTH.OBJ.

GothicFontProc est le nom de la procédure employée dans le programme

d'implantation (et non pas "implémentation"...) **FONTS.PAS**. Comme on emploie ce programme tel qu'il est, nous ne touchons pas aux noms qu'il impose.

Si vous avez bien entré la ligne ci-dessus, vous pouvez vérifier qu'on a bien créé un programme GOTH.OBJ (avec DIR...).

On fait de même avec les autres fontes :

BINOBJ LITT.CHR LITT SmallFontProc (small = little = petit)
BINOBJ TRIP.CHR TRIP TriplexFontProc
BINOBJ SANS.CHR SANS SansSerifFontProc (attention aux s et f)

Une fois cette manipulation terminée, vous avez vos quatre fichiers *.OBJ.

On va les regrouper et en faire une unité. Pour cela, revenez (par EXIT) sous Turbo Pascal, et chargez le programme FONTS.PAS.

Vous pouvez examiner ce programme : c'est un bon exemple de création d'unité. Ce n'est quand même pas notre propos dans ce livre. Contentons-nous de l'utiliser. Nous allons le compiler SUR DISQUE. Et non pas en mémoire comme on le fait habituellement (c'est un fichier qui ne peut pas être "runé"). Donc vous pressez ***<Alt>-C***, descendez sur l'indication *Compilation memory* (compilation en mémoire), pressez ***<Enter>*** pour qu'elle devienne *Compile to disk*, remontez sur compile et pressez *<Enter>*. Bruit de lecteur. Tout va bien, sauf si vous avez fait une erreur en écrivant SansSerifFontProc...

Un coup d'oeil sur le catalogue : merveille, on a bien un fichier FONTS.TPU qui est votre première (sûrement pas la dernière) unité en Turbo Pascal.

Repassez sous MS-DOS, et éliminez les fichiers désormais inutiles et encombrants par ***DEL *.CHR*** et ***DEL *.OBJ***. Ils sont remplacés par ***FONTS.TPU*** pour la suite.

Vous faites maintenant une opération similaire avec les "drivers".

BINOBJ CGA.BGI CGA CGADriverProc
BINOBJ EGAVGA.BGI EGAVGA EGAVGADriverProc
BINOBJ HERC.BGI HERC HercDriverProc
BINOBJ ATT.BGI ATT ATTDriverProc
BINOBJ PC3270.BGI PC3270 PC3270DriverProc

Puis retour sous Turbo Pascal et compilation sur disque (on est déjà sur la bonne option) de DRIVERS.PAS. Retour sous MS-DOS pour vérifier que DRIVERS.TPU est présent (on vous avait bien dit que FONTS.TPU n'était pas votre dernière Unité !), et pour supprimer les fichiers *.OBJ, *.CHR, et, tant qu'à faire (la place nous est comptée) :

BINOBJ.EXE, DRIVERS.PAS et FONTS.PAS.

La disquette est propre, et ne contient plus grand'chose...

Maintenant, il faut faire comprendre à notre programme LISS2 (écrit à partir de LISS1) qu'il faut utiliser ces *.TPU. Bien sûr, des *Uses Drivers* et *Uses Fonts* en début de programme sont nécessaires. Mais pas suffisant. Il faut donner au programme l'ordre, lors de la compilation, d'intégrer ces unités en son sein. Et vérifier que tout se passe bien. Une seule voie sûre pour nous : dire à Turbo Pascal d'enregistrer le code intégré correspondant au gestionnaire ou à la police de caractères, ceci grâce aux fonctions ***RegisterBGIdriver*** et ***RegisterBGIfont***. Fonctions, avons-nous dit : elles retournent en effet un code négatif si le résultat est incorrect, et le numéro interne de gestionnaire ou de police si tout va bien. Il va de soi que si ça va mal, on arrête les frais avec ***Halt(1)***.

Ces fonctions ***Registerxxx*** utilisent des pointeurs, sans doute parce que ce sont les mêmes qu'en Turbo C... D'où le "aronba" qui précède les noms des *xxxProc*. Quand vous plongerez-vous dans les délices des pointeurs ?

La procédure ***PrendLesUnits*** correspondante est plutôt pénible à frapper. Il est vrai que c'est la dernière fois que vous le faites : on peut ensuite la recopier directement !

Vous avez ci-après le programme complet de LISS2. On aurait pû se contenter de ne mettre que le début (jusqu'à la fin de la procédure *PrendLesUnits*) et la fin (le "corps" du programme) : il n'y a que là qu'il y ait des modifications par rapport à LISS1. Vous n'avez en fait que bien peu de frappe à faire !

Maintenant, un petit jeu : compilez LISS2 sur disque (vous l'avez peut-être fait par inadvertance, puisqu'on n'a pas dit de repasser sur compilation en mémoire...). Et essayez le programme LISS2.EXE, qui "pèse" plus de 74 Ko, sur tous les ordinateurs PC qui vous tombent sous la main. Bon test de compatibilité...

Nous avons :

- utilisé diverses fontes de caractères ;
- créé une Unité pour les fontes ;
- créé une Unité pour les gestionnaires d'écran ;
- créé un programme de plus de 64 Ko ;
- utilisé des intructions graphiques complexes.

```
program LISS2;          (* en hommage à Lissajous.
             Ce programme trace des "ellipses" encadrées par
             des rectangles en différentes couleurs et tailles. *)

(* Ce deuxième programme intégre les fontes
      et les gestionnaires d'écran.
   Il affiche un bref message en différentes
      tailles et caractères.                *)

Uses Crt,
     Graph,
     Fonts,
     Drivers;

var ParamCos, ParamSin : real;          (* définies par auteur *)
    Couleur, MaxCouleur,                (*       idem          *)
    AmplitudeX, AmplitudeY,
    DemiLargeur, DemiHauteur,
    QuartLargeur, QuartHauteur : word;
    GraphDriver, GraphMode : integer;  (* noms copiés sur Borland *)

(*-------------------------------------------------------------------*)

procedure PrendLesUnits;
begin
            (* mise en place des drivers *)
     if (RegisterBGIdriver(@CGADriverProc)<0) or
      (RegisterBGIdriver(@EGAVGADriverProc)<0) or
      (RegisterBGIdriver(@HercDriverProc)<0) or
      (RegisterBGIdriver(@ATTDriverProc)<0) or
      (RegisterBGIdriver(@PC3270DriverProc)<0) then Halt(1);

            (* mise en place des fontes *)
     if (RegisterBGIfont(@GothicFontProc)<0) or
      (RegisterBGIfont(@SansSerifFontProc)<0) or
      (RegisterBGIfont(@SmallFontProc)<0) or
      (RegisterBGIfont(@TriplexFontProc)<0) then Halt(1);
end;

(*-------------------------------------------------------------------*)
```

```
procedure InitialiseEcran;

(* trouve quel est le gestionnaire d'écran en place et
   prend le mode couleurS en CGA. Tous "drivers" utilisables. *)

begin
     DetectGraph(GraphDriver,GraphMode);
     Case GraphDriver of
          CGA : GraphMode := Random(4);   (* refuse donc le mode 4 *)
          MCGA : GraphMode := Random(4);  (*          idem         *)
          EGA : GraphMode := 1;
          VGA : GraphMode := 2  (* rappel : pas de ';' avant else *)
          else GraphMode := 0;
          end;
     InitGraph(GraphDriver,GraphMode,'');
end;

(*-------------------------------------------------------------------*)
procedure Parametres;   (* calcul des paramètres de "l'ellipse" *)
begin

     (* paramètres de l'écran, imposés par le driver et le mode *)
     MaxCouleur := GetMaxColor;
     DemiLargeur := GetMaxX div 2;     (* de l'écran, en pixels *)
     DemiHauteur := GetMaxY div 2;
     QuartLargeur := DemiLargeur div 2;
     QuartHauteur := DemiHauteur div 2;

     (* paramètres aléatoires : *)

     (* paramètres pour la courbe *)
     ParamCos := 1 + (Random(30)/(1 + Random(5)) ) ;
     ParamSin := 1 + (Random(30)/(1 + Random(5)) ) ;
     Couleur := 1 + Random(MaxCouleur);

     (* paramètres généraux *)
     AmplitudeX := QuartLargeur + Random(QuartLargeur);
     AmplitudeY := QuartHauteur + Random(QuartHauteur);
end;

(*-------------------------------------------------------------------*)

function CalcX(z : integer) : word;
(* cette fonction retourne la valeur de X *)
begin
     CalcX := DemiLargeur + Trunc(AmplitudeX * cos(z*ParamCos));
end;
(*-------------------------------------------------------------------*)
```

```
function CalcY(z : integer) : word;
(* celle-ci la valeur de Y ... *)
begin
     CalcY := DemiHauteur + Trunc(AmplitudeY * sin(z*ParamSin));
end;

(*-------------------------------------------------------------------*)

procedure Trace;
(* trace le rectangle et la courbe de Lissajous qui
   correspondent aux valeurs calculées.          *)

(* variables qu'il n'est pas utile de déclarer globalement : *)
var RectHGX, RectHGY, RectBDX, RectBDY : word;
    i : integer;

begin
     (* tracé du rectangle *)
     RectHGX := DemiLargeur - AmplitudeX - 1;
     RectHGY := DemiHauteur - AmplitudeY - 1;
     RectBDX := DemiLargeur + AmplitudeX + 1;
     RectBDY := DemiHauteur + AmplitudeY + 1;
     SetColor(1 + Random(MaxCouleur));   (* couleur tracé rectangle *)
     Rectangle(RectHGX,RectHGY,RectBDX,RectBDY);

     (* tracé de la courbe de Lissajous *)
     for i:=1 to 2000 do PutPixel(CalcX(i),CalcY(i),Couleur);

end;

(*-------------------------------------------------------------------*)

procedure Ecriture;
begin
     SetTextJustify(CenterText,CenterText);
     SetTextStyle(Random(5),HorizDir,1+Random(10));
     SetColor(1 + Random(MaxCouleur));
     OutTextXY(DemiLargeur, DemiHauteur, 'P.S.I.');
end;

(*===================================================================*)
```

```
begin
     Randomize;
     PrendLesUnits;
     while not KeyPressed do
          begin
               InitialiseEcran;
               Parametres;
               Ecriture;
               Trace;
          end;
     CloseGraph;
     RestoreCrtMode;
end.

(*FFFF
```

CHAPITRE 12

DEUXIEME RECAPITULATION

Ce qui suit est un résumé ordonné de ce que nous avons vu, avec quelques mises au point et quelques compléments.

ELEMENTS DE BASE DU LANGAGE

Symboles de base

Signes : lettres, chiffres et caractères divers (parenthèses, crochets, etc.).

Identificateurs : formés par une combinaison de lettres et de chiffres sans espace avec emploi possible du signe "_". Majuscules ou minuscules (Turbo Pascal ne fait pas la différence) non accentuées. Ne pas commencer par un

chiffre. Jusqu'à 63 signes pour un identificateur : ne pas se gêner pour prendre des noms explicites !

Opérateurs : - pour l'affectation, c'est le double signe :=

- pour les relations ce sont les symboles habituels :
  = < > <= >= <>

- pour la logique : AND, OR, XOR.

Séparateur : essentiellement le point-virgule.

Symboles spéciaux : les parenthèses, les crochets (pour les tableaux et les chaînes), les accolades (pour les commentaires qui peuvent aussi être mis entre "(*" et "*)") et pour certaines instructions particulières).

Mots réservés

Ce sont les mots connus de Turbo Pascal pour des emplois très stricts. Ne pas les mélanger avec d'autres, c'est-à-dire, par exemple ne pas créer une variable du nom de "ordinateur" qui commence par le mot réservé "or" (risque de conflit), et surtout ne pas prendre un mot réservé pour un emploi autre que celui prévu.

La liste ci-dessous est exhaustive. En gras, ceux qui apparaissent dans ce livre, en tout petit un mot obscène (pour tout programmeur bien élevé) qu'il nous a bien fallu mettre, mais qui n'est pas vraiment du Pascal.

absolute	**file**	mod	shr
and	**for**	nil	**string**
array	forward	**not**	**then**
begin	**function**	**of**	**to**
case	goto	**or**	**type**
const	**if**	packed	**unit**
div	implementation	**procedure**	**until**
do	in	**program**	**uses**
downto	inline	**record**	**var**
else	interface	**repeat**	**while**
end	interrupt	**set**	**with**
external	label	shl	**xor**

CONSTITUTION D'UN PROGRAMME

En-tête : la mention *program Nom-du-Programme* est facultative en Turbo Pascal.

Emploi d'Unités : les Unités de Turbo Pascal version 4 sont des sortes de banques d'instruction particulières, éventuellement créées par le programmeur. Elles doivent être signalées dès le début du programme par la mention : *Uses Unit1, Unit2, Unit3,... ;*

Si une unité en utilise une autre, cette dernière doit être signalée avant dans la mention *Uses* ci-dessus.

Déclarations

Etiquettes : pour mémoire, *label* permet de déclarer des "sous-programmes" et d'employer *goto.* Beurk !

Constantes : peuvent être données directement.
Exemple :
```
const e = 2.718;
   titre = 'Turbo Pascal';
```

Types : voir ci-après les différents types standards et les modes de déclaration par l'utilisateur.

Variables : lorsqu'elles sont de type standard, le type n'a pas besoin d'être déclaré au préalable.

Procédures

Commencent par un identificateur, puis *begin* et *end;* encadrent le corps de ce qui est, d'une certaine façon, une instruction nouvelle.

Une procédure peut contenir des déclarations de ses propres variables (variables dites locales, par différence avec celles ci-dessus qui sont globales) qui n'ont d'existence que dans cette procédure.

Fonctions

Ressemblent à une procédure. Renvoient une valeur.

TYPES DE VARIABLES

Types standards simples

TYPES ORDONNES

Ce sont ceux sur lesquels on peut définir un ordre (la numérotation peut être retrouvée par la fonction *Ord*).

ENTIERS

shortint : de - 128 à 127. Occupe un octet.
integer : de - 32 768 à 32 767 (MaxInt). Occupe deux octets.
longint : de - 2 147 483 648 à 2 147 483 647. Quatre octets.
byte : de 0 à 255. Un octet.
word : de 0 à 65 535. Deux octets.

BOOLEEN

Il n'y a que deux "valeurs" : TRUE et FALSE. Remarquez que FALSE <TRUE, que *Ord* (FALSE) vaut 0 (et 1 pour *Ord* (TRUE) ...), que *Succ* (False) = TRUE de même que *Pred* (TRUE) = FALSE.

CARACTERE

C'est le type **char**.

L'ordre est celui de la table ASCII. Par exemple *Ord(A) = 65* et on peut écrire que 6 < g...

Un caractère peut aussi être considéré comme une chaîne de longueur 1.

ENUMERATION

En fait, une succession de constantes qu'on déclare en fixant l'ordre.
Exemple : *couleurs = (pique, coeur, carreau, trefle)* avec ici *pique < carreau, Ord (coeur) = 1, etc.*

Pour mémoire : type intervalle (défini à partir d'un des types ci-dessus).
Exemple : *minuscule = 'a'..'z';*

REEL

Ce sont les "nombres à virgules", et non pas les nombres réels des mathématiciens (pour être plus précis, on a affaire à des sous-ensembles de l'ensemble des nombres décimaux...). Il paraît en plus qu'en informatique, cette virgule, qui d'ailleurs est un point, flotte. Sur quoi ? Passons. La notation peut être aussi bien un "classique " 0.0034502 que la notation 3.4×10^{-3} que 34×10^{-4} (qu'a fait la "virgule-point" pendant ce temps ?).

Si on n'a pas de co-processeur de calcul, ce qui est très probablement votre cas, on se contente du seul type **real** qui permet d'aller de -2.9×10^{39} à 1.7×10^{38} (des centaines de milliards de milliards de milliards de milliards...) avec 11 chiffres significatifs. Prend six octets.

Chaînes et tableaux

La déclaration de type n'est pas obligatoire.

Dans les deux cas, on peut repérer un élément par sa position.

CHAINES

Une chaîne de caractères (string) est définie par sa longueur :

var Nom : string[30];

Conseil : si plusieurs chaînes servent de la même manière (et, bien sûr, ont la même longueur) faire d'abord une déclaration de type qui permet au lecteur de mieux percevoir qu'elles sont semblables.

Exemple :

```
type Message = string[200];
var Annonce, Memo, Billet, PenseBete : message;
```

TABLEAUX

Un tableau de variables est déclaré avec la numérotation de son contenu.

var Affichage : Array[1..10] of Billet;

pour un tableau de 10 messages (de 200 signes au plus) numérotés de 1 à 10.

On peut déclarer des tableaux de tableaux :

```
type panneau = Array[2..5] of Array[1..10] of Annonce;
```

ou, plus directement :

```
type panneau = Array[2..5,1..10] of Annonce;
```

Il y a le choix. Se laisser guider par le contexte.

Autres types

RECORD

Appelé "enregistrement". Pensez à "dossier" (qui peut, bien sûr, être enregistré !).

Un *record* est fait de plusieurs composantes (ou "champs") de types divers.

Exemple :

```
type  UnMois = (Janvier,Fevrier,Mars,Avril,Mai,Juin,Juillet,Aout,
                Septembre,Octobre,Novembre,Decembre);

      Date = Record
                Jour : 1..31;
                Mois : UnMois;
                Annee : integer;
        end;

var CeJour : Date;
```

On pourra, dans le programme, écrire :

```
with CeJour do
    begin
      Jour := 4;
      Mois := Mars;
      Annee := 1988;
    end;
```

Le type "enregistrement" est extrêmement commode. Ne le négligez pas, ce n'est pas un gadget !

SET : c'est le type "ensemble ordonné", avec réunion, intersection, etc...

FILE : fichier. C'est tout simplement une série de composantes d'un type déterminé...

Exemple : var Notes : file of byte;

POINTER : pointeur, mis ici pour dire qu'il existe. Trop compliqué pour l'instant !

OPERATEURS

Un opérateur renvoie toujours une valeur.

Ceux que nous avons vu sont :

- ceux de l'arithmétique : + - * / mais aussi *div* (division entière). Il y en a d'autres : *mod* (reste de la division entière), et toute une série pour le travail sur les bits...
- ceux de la logique : travaillent avec les deux booléens. Reprendre ce qui est vu au chapitre 9.
- ceux pour les comparaisons : il y a bien une valeur "renvoyée" même si cela reste implicite.

  Exemple : *(3 > 10)* renvoie *False*...

INSTRUCTIONS ELEMENTAIRES

Conditionnelles

if... then... else...

Beaucoup moins facile à utiliser qu'on ne le pense, car elle incite à des imbrications vite inextricables.

Syntaxe très simple : **si** telle condition est réalisée **alors** on fait telle action **sinon** on fait telle autre.

On peut imbriquer : **exemple :**

```
if a<b then
   if c>d then begin
```

```
    maproc;
    if d=e then saproc
    else autreproc;
    if d>c saproc else if a<d encorune;
  end
  else if      etc. Pas de commentaire supplémentaire !
```

case... of... else

Permet de bien séparer les différents cas. On peut utiliser des intervalles aussi bien que des valeurs précises. Instruction claire.

Exemple :

```
case Vitesse of
     0..59 : Write('Agglomération.');
     60..89 : Write('Possible sur route.');
     90..109 : Write('Possible sur deux fois deux voies.');
     110..129 : Write('Autorisé sur autoroute.');
     130..MaxInt :Write('Seulement sur circuit.');
end;
```

Répétitives

Elles sont trois qui se ressemblent, avec cependant des différences importantes. Elles font s'exécuter une instruction en vérifiant à chaque exécution si certaine condition est vraie.

Une seule des trois fait exécuter (une fois) avant de vérifier : c'est *repeat... until...*

while...do... vérifie si la condition est vraie avant l'exécution.

for...to... ne fonctionne qu'avec une condition numérique qu'elle sait incrémenter (ou décrémenter) avec un "pas" donné par le programmeur.
Exemple : *for i:=10 to 1 downto 2 do...*

Bien s'assurer que la condition de fin peut être atteinte ! Dans l'exemple ci-dessus, la boucle est infinie : les valeurs possibles pour *i* sont 10, 8, 6, 4, 2, 0, -2 etc. mais pas 1...

Eviter aussi d'imbriquer les boucles. Sauf bien sûr si vous faites un concours d'herméticité, auquel cas vous pouvez aussi ajouter des imbrications de *if* et des *goto*.

PROCEDURES ET FONCTIONS STANDARDS

Ce sont celles qui appartiennent à tous les langages Pascal. Nous avons vu (plus ou moins rapidement) celles qui sont en gras :

Contrôle de l'exécution :

Exit	fait sortir d'une routine (cf goto...).
Halt	arrête l'exécution.

Gestion de la mémoire :

Dispose	FreeMem	GetMem	Mark
MaxAvail	MemAvail	New	Release

Transformation d'un type en un autre :

Chr	retourne le caractère de numéro indiqué.
Ord	retourne le numéro de la variable.
Round	arrondit un réel en l'entier long le plus proche.
Trunc	prend la partie entière d'un réel.

Mathématiques :

Abs	valeur absolue de...
ArcTan	arc tangente...
Cos	cosinus
Exp	exponentielle de
Frac	retourne la partie non entière
Int	retourne la partie entière
Ln	logarithme népérien
Pi	3.141592653587932385
Sin	sinus
Sqr	carré
Sqrt	racine carrée

Sur les variables ordonnées :

Dec	décrémentation
Inc	incrémentation
Odd	test de parité
Pred	donne la précédente
Succ	donne la suivante

Sur les chaînes de caractères :

Concat	Copy	Delete	Insert
Length	Pos	Str	Val

Pointeurs et adresses :

Addr	CSeg	Dseg	Ofs
Ptr Seg	Sptr	SSeg	

Divers :

FillChar	Hi	Lo	Move
ParamCount	ParamStr	**Random**	**Randomize**
SizeOf	Swap	**Upcase**	

Vous constatez que nous n'avons vu qu'une toute petite partie de ces procédures et fonctions. Vous n'aurez que rarement besoin de la plupart d'entre elles : ce sont des instructions très "près" de la machine qui sont d'un emploi délicat. En général, vous utiliserez des fonctions et procédures préparées par Borland, telles que *Write* ou *GotoXY*...

Liste des fonctions et procédures du manuel

En gras, celles que nous avons utilisées. Il y en a beaucoup d'autres, comme vous pouvez en juger !

Abs	Addr	Append	Arc
ArcTan	**Assign**	AssignCrt	Bar
Bar3D	BlockRead	BlockWrite	ChDir
Chr	Circle	ClearDevice	ClearViewPort
Close	**CloseGraph**	ClrEol	**ClrScr**
Concat	Copy	**Cos**	CSeg
Dec	**Delay**	Delete	DelLine
DetectGraph	DiskFree	DiskSize	Dispose
DosExitCode	DrawPoly	DSeg	Ellipse
Eof	Eoln	Erase	Exec
Exit	Exp	FilePos	**FileSize**
FillChar	FillPoly	FindFirst	FindNext
FloodFill	Flush	Frac	FreeMem
GetArcCoords	GetAspectRatio	GetBkColor	GetColor
GetDate	GetDir	GetFAttr	GetFillPattern
GetFillSettings	GetFTime	**GetGraphMode**	GetImage
GetIntVec	GetLineSettings	**GetMaxColor**	**GetMaxX**
GetMaxY	GetMem	GetModeRange	GetPalette
GetPixel	GetTextSettings	**GetTime**	GetViewSettings
GetX	GetY	**GotoXY**	GraphDefaults
GraphErrorMsg	GraphResult	**Halt**	Hi
HighVideo	ImageSize	**Inc**	**InitGraph**
Insert	InsLine	Int	Intr
IOResult	Keep	**KeyPressed**	Length
Line	LineRel	LineTo	Ln
Lo	**LowVideo**	Mark	MaxAvail

MemAvail	MkDir	Move	MoveRel
MoveTo	MSDos	New	**NormVideo**
NoSound	**Odd**	Ofs	**Ord**
OutText	**OutTextXY**	PackTime	ParamCount
ParamStr	Pi	PieSlice	Pos
Pred	Ptr	PutImage	**PutPixel**
Random	**Randomize**	**Read**	**ReadKey**
ReadLn	**Rectangle**	**RegisterBGIdriver**	**RegisterBGIfont**
Release	Rename	**Reset**	RestoreCrt
RestoreCrtMode	**ReWrite**	RmDir	Round
Seek	SeekEof	SeekEoln	Seg
SetActivePage	SetAllPalette	SetBkColor	**SetColor**
SetDate	SetFAttr	SetFillPattern	SetFillStyle
SetFTime	SetGraphBufSize	SetGraphMode	SetIntVec
SetLineStyle	SetPalette	SetTextBuf	**SetTextJustify**
SetTextStyle	SetTime	SetUserCharSize	SetViewPort
SetVisualPage	**Sin**	SizeOf	Sound
SPtr	Sqr	Sqrt	SSeg
Str	Succ	Swap	TextBackground
TextColor	TextHeight	TextMode	TextWidth
Trunc	Truncate	UnPackTime	**UpCase**
Val	WhereX	WhereY	Window
Write	**WriteLn**		

CONCLUSION

Vous avez fait un premier voyage au pays de la version 4 de Turbo Pascal. Déjà, vous en savez assez pour écrire bon nombre de programmes efficaces et rapides.

Mais il reste encore beaucoup à apprendre ! Arriver à se retrouver dans les nombreuses procédures et fonctions qui vous sont offertes sur vos disquettes n'est pas le plus difficile, de même que travailler sur les chaînes de caractères.

Ce qu'il vous faudra surtout étudier, c'est d'abord comment se fait le passage des paramètres d'une procédure à une autre (par valeur, par variable). Puis les pointeurs, qui vous permettront de créer des procédures et fonctions beaucoup plus puissantes. Il vous faudra apprendre comment utiliser au mieux la mémoire de votre ordinateur. Et, bien sûr, comment agir avec les bits de cette mémoire...

Votre manuel Borland donne par lui-même beaucoup d'indications. Il est certain qu'il y aura bientôt une abondante liste d'ouvrages traitant de cette version 4 de Turbo Pascal. Vous avez de quoi vous occuper...

Profitez bien de Turbo Pascal version 4 !

ANNEXE 1

PREPARATION DES DISQUETTES

Si vous êtes l'heureux utilisateur d'un disque fixe, vous pouvez, dans un premier temps, mettre tous les fichiers des trois disquettes dans le même répertoire. Quand vous aurez un peu plus d'habitude, il vaudra mieux créer des sous-répertoires spécifiques.

Ce livre est exploitable si votre ordinateur ne possède qu'un seul lecteur de disquettes. Il vous faut quatre disquettes :

- une disquette "système",
- trois disquettes "programme".

Il faut d'abord formater (préparer physiquement) ces disquettes. Pour ce faire, allumer votre ordinateur, mettre sa disquette "système", presser une touche, attendre un peu que le signe **A>** apparaisse, puis taper **FORMAT A:** et <Return> (ou <Enter>). Suivre alors les indications données à l'écran. Le faire pour les quatre disquettes que vous étiquettez "SYSTEME", "TP1", "TP2", "TP3".

Il faut ensuite remettre en place la disquette "système" (que vous n'utiliserez plus ensuite) : on va en faire une copie. La commande de recopie d'un disque est : **DISKCOPY A:** et <Return> bien sûr. Là encore, suivre les indications à l'écran. La disquette "SYSTEME" que vous venez de créer vous servira désormais pour tout lancement.

Normalement, si vous n'avez qu'un seul lecteur de disquettes, votre programme de lancement prévoit de créer un disque virtuel, tout en mémoire. Vérifiez-le par : **DIR C:** qui vous renverra la taille de ce disque fictif. S'il n'y a rien, allez vous expliquer avec votre revendeur...

Mettez alors la disquette 1 de Turbo Pascal dans votre lecteur et tapez : **COPY TURBO.EXE C:** et <Enter> bien entendu...

Vous remplacez cette disquette par celle marquée TP1 et faites l'opération inverse : **COPY C:TURBO.EXE A:** et <Enter>. Même chose avec TP2 et TP3 (il suffit de presser la touche de fonction <F3> pour que l'ordre ci-dessus réapparaisse).

On libére la mémoire par **DEL C:*.EXE**

Copie de TURBO.TPL vers C: de la même manière, et de C: vers TP1, TP2, TP3.

On procéde de même avec GRAPH.TPU, à la différence qu'il est lui sur la disquette 3 de Turbo Pascal.

En ce qui concerne les fichiers en BGI : ne prenez que le "vôtre" (celui qui correspond à votre écran) sur les disquettes TP1 et TP2 : ce n'est pas la peine de les surcharger. Si vous avez des doutes, mettez-les tous, comme indiqué ci-dessous.

Pour charger d'un coup tous les fichiers de gestion d'écran sur la disquette TP3, il suffit d'entrer **COPY *.BGI C:** puis, après échange de disquettes, **COPY C:*BGI A:**

Même chose avec les fichiers de fontes (uniquement sur la disquette TP3) : **COPY *.CHR C:** puis **COPY C:*CHR A:**

ANNEXE 2

COMMANDES DE L'EDITEUR

Voir tableau page suivante.

Commandes de l'éditeur

SORTIE DE L'EDITEUR : Ctrl-K-D

DEPLACEMENT CURSEUR

Ligne au-dessus	: **Ctrl-E**	Remonte d'une page	: **Ctrl-R**
Ligne au-dessous	: **Ctrl-X**	Descente d'une page	: **Ctrl-C**
Colonne de gauche	: **Ctrl-S**	Début du mot suivant	: **Ctrl-F**
Colonne de droite	: **Ctrl-D**	Début mot précédent	: **Ctrl-A**
Première colonne	: **Ctrl-Q-S**	Fin de ligne	: **Ctrl-Q-D**
Haut de l'écran	: **Ctrl-Q-E**	Bas de l'écran	: **Ctrl-Q-X**
Début du fichier	: **Ctrl-Q-R**	Fin du fichier	: **Ctrl-Q-C**
Dernière position	: **Ctrl-Q-P**	Tabulation (en général)	: **<TAB>**

DEROULEMENT d'une ligne : vers le bas : **Ctrl-W**
vers le haut : **Ctrl-Z**

RATTRAPAGE : annuler les modifications de la ligne que le curseur n'a pas encore quittée : **Ctrl-Q-L**

SUPPRESSION

Caractère sous le curseur	: **Ctrl-G**
Mot à droite du curseur	: **Ctrl-T**
Ligne du curseur (danger!)	: **Ctrl-Y**
Fin de la ligne du curseur	: **Ctrl-Q-Y**
Sur la plupart des ordinateurs	: caractère à gauche du curseur : **<DEL>**

INSERTION : le mode normal est l'insertion.
Bascule insertion/recouvrement : **Ctrl-V**
Insertion d'une ligne : **Ctrl-N**
Moins nécessaire :
pour la position du curseur après <RETURN> : **CTRL-Q-I**

COMMANDES DE BLOC : un bloc est une partie de texte...
Marquer le début d'un bloc : **Ctrl-K-B** Marquer la fin : **Ctrl-K-K**
Recopie sans enlever : **Ctrl-K-C** **En enlevant** : **Ctrl-K-V**
Supprime le bloc (danger!) : **Ctrl-K-Y**
Prend un fichier sur disque : **Ctrl-K-R** Met sur disque : **Ctrl-K-W**

ANNULATION d'une opération en cours quand une entrée est demandée : **Ctrl-U**

CARACTERE DE CONTROLE DANS UN TEXTE : à faire précéder de **Ctrl-P**

ANNEXE 3

"JEU DE LA VIE"

Quelques références : *Le Monde* du 30/10/74.
Science et Vie de juillet 72, janvier et mars 75.
Jeux et Stratégie n° 12, juin/juillet 81.
Pour la Science d'avril 87 (dans l'espace).

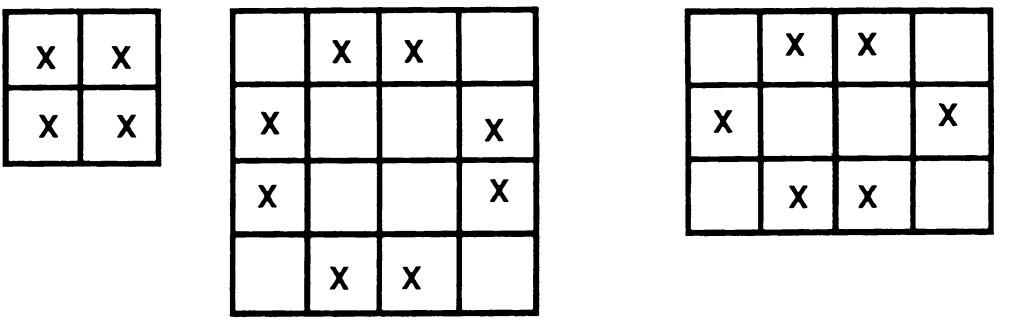
Historique : Dans les années 40, John von Neumann imagina un "automate cellulaire", c'est-à-dire un damier où des pions vivent et meurent suivant des règles logiques, en fonction de leur voisinage.

En 1968, John Horton Conway inventa le "jeu de la vie", celui que nous avons utilisé, nomenclaturé depuis VIE 2333. Cette numérotation est à décoder par paires des chiffres : une cellule reste vivante si elle a deux ou trois voisines et un espace vide, voit une naissance si il a au plus trois et au moins trois cellules vivantes voisines (exactement trois, donc).

Gros succès au début des années 70 ! Le plus souvent, on joue sur une feuille avec un crayon... admirez le courage et la persévérance ! Quelques chanceux utilisent des "gros" ordinateurs de l'époque.

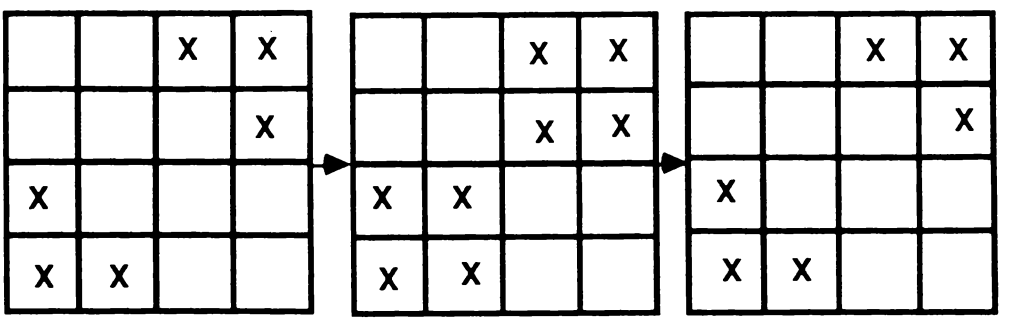
Une des idées principales est de trouver des populations particulières. Dans tous les exemples suivants, l'environnement est vide.

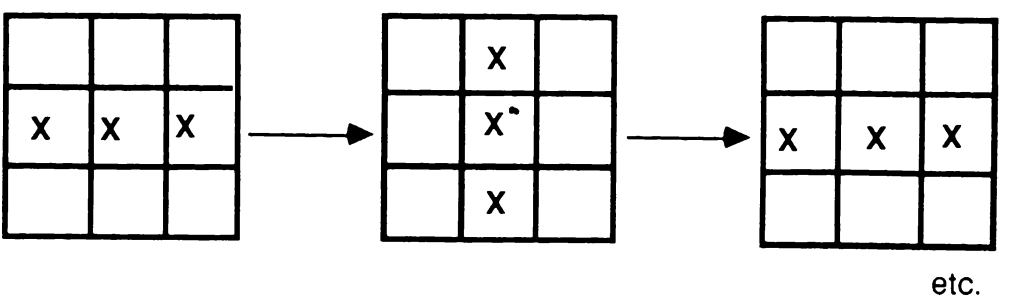
- **Totalement stables** :



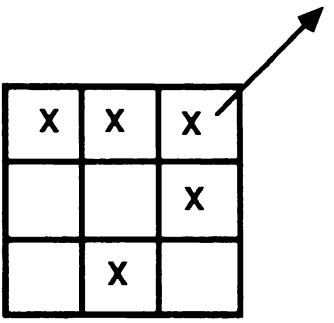
Il y en a d'autres !

- **Périodiques** :





- **Mobiles** : le "planeur" :



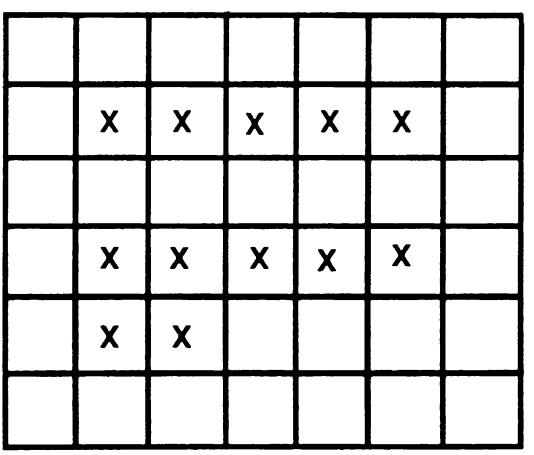
- **Expansives** : qui accroissent indéfiniment le nombre de leurs cellules. La méthode que nous employons ici n'est guère propice à une telle recherche, du fait des "effets de bord".

- **Reproductrices** : qui engendrent des populations identiques à elles-mêmes, en se retrouvant intactes... en trouverez-vous ?

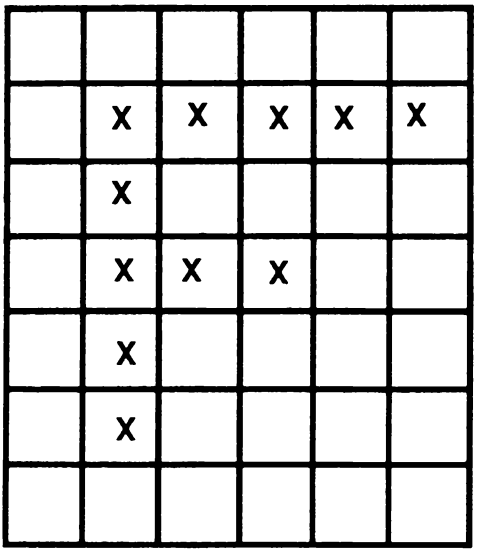
- **Orphelines** : qui n'ont pas d'antécédent. Pas faciles à trouver ! La plus simple que nous connaissions est contenue dans un rectangle de 9 x 33...

Tout le "jeu" consiste donc à trouver des configurations de départ qui donnent des résultats particuliers.

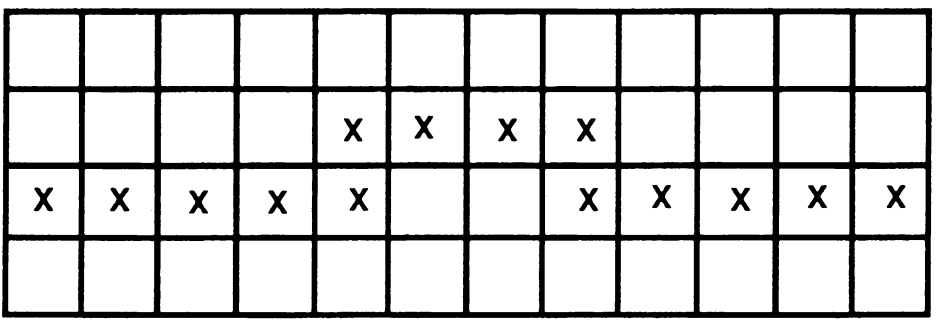
Exemples :



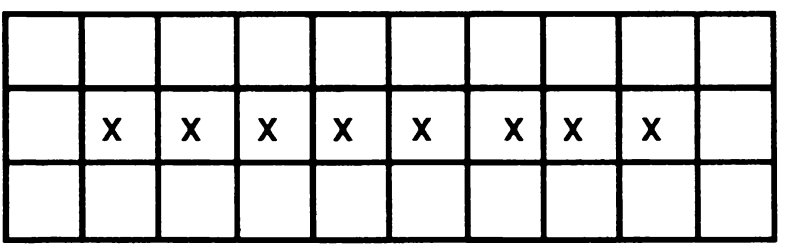
finit par "déborder" de l'écran



devient stable à la huitième génération.

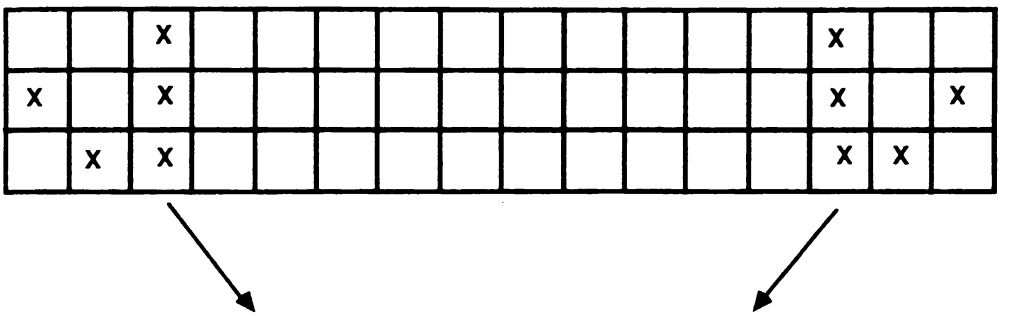


est stable au bout de six générations.

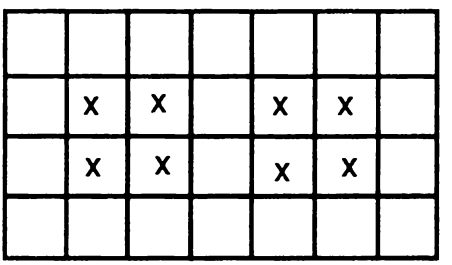


n'est stable qu'au bout de 51 générations.

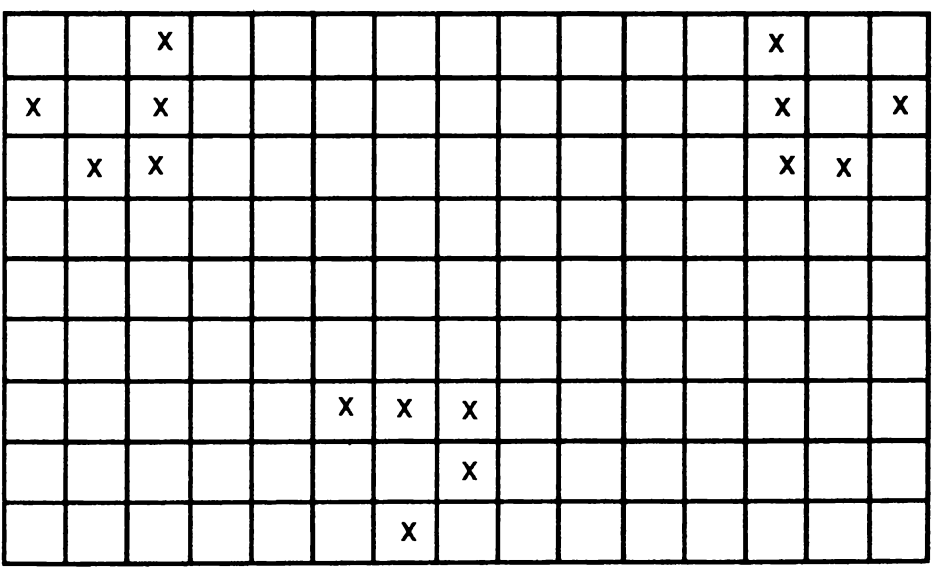
On peut aussi examiner la collision de "planeurs" :

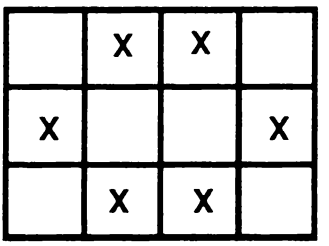


va donner (décalé vers le bas de l'écran), au bout de 24 générations, la configuration stable :



Avec trois planeurs, des résultats variés suivant les emplacements de base. Ainsi, la population initiale ci-dessous se stabilise au bout de 66 générations :





(stable)

Autre possibilité : bien surveiller l'évolution de la population en marquant différemment chaque génération de nouvelles cellules. Ainsi, la première génération serait indiquée par des "a", la deuxième par des "b", etc. Dans le cadre de notre programme, c'est assez facile tant qu'on ne cherche pas à dépasser un nombre relativement faible de générations.

On peut jouer aussi à plusieurs. Le plus simple étant à deux. Exemple de "règle" : chaque joueur dispose, à tour de rôle, de cinq cellulles, avec sa couleur (ou son signe) personnel. Une cellule qui naît prend la couleur majoritaire de ses trois parentes. A gagné celui qui obtient le plus de cellules au bout d'un nombre donné de générations. On peut faire une variante en ajoutant des cellules toutes les cinq ou dix générations... Beaux programmes en perspective !

En ce qui concerne le jeu dans l'espace, où chaque cellule peut avoir 26 voisines, il y a plusieurs possibilités. D'après A. Dewdney, qui signe l'article de *Pour la Science* d'avril 1987 cité ci-dessus, les types VIE 4555 et VIE 5766 ressemblent beaucoup à notre VIE 2333. La plus grande difficulté résulte de la représentation dans l'espace.

ANNEXE 4

CODES CLAVIER ETENDUS

Ce sont ceux de toutes les touches qui ne sont pas des caractères...

A chaque fois, c'est un code double, le premier code étant toujours zéro. Le deuxième est donné, en décimal, par la table ci-dessous, qui n'est pas exhaustive (problèmes de configurations nationales qui peuvent redéfinir certaines touches) :

<shift> <TAB>	15
F1 à F10	59 à 68
<Home>	71
Flèche haut	72
Page haut	73 (c'est <PgUp> assez souvent...)
Flèche gauche	75
Flèche droite	77
<End> (Fin)	79
Flèche bas	80
Page bas	81 (ou encore <PgDn>)

Insertion	82 (<Ins>)
Destruction	83 (aussi dénommée <Del>, <Eff>, etc.)
<Shift> F1 à F10	84 à 93
<Ctrl> F1 à F10	94 à 103
<Alt> F1 à F10	104 à 113
<Ctrl> Prt Sc	114
<Ctrl> flèche gauche	115
<Ctrl> flèche droite	116
<Ctrl> <End>	117
<Ctrl> <PgDn>	118
<Ctrl> <Home>	119
<Crtl> 1 à 9, 0, -, =	120 à 131
<Ctrl> <PgUp>	132

Il peut être plus lisible d'utiliser ces codes en les traduisant en lettres. Nous ne redonnons pas ici des codes ASCII, qui se trouvent dans tous les manuels...

INDEX

AND 74
Array 63
Assign. 81

begin 19
Blink 44
byte 54

Case...of... 45,55
char 28,54
Close 81
CloseGraph 98
ClrScr 23
compilation en mémoire . 23
" sur disque . 13
Crt 20

définition (affichage) 91
Delay 29
DetectGraph 96
div 97
do 19
Dos (unité) 34

éditeur 14,126
end 18,19

EXE 13
Exécution d'un programme . 13

False 57
FileSize 82
for...to... 42,55

GetMaxColor 97
GetMaxX 97
GetMaxY 97
GetTime 33
GotoXY 41,57

HighVideo 42

if...then... 44,55
Inc 27
indice 42
InitGraph 97

LongInt 38
LowVideo 42

mode d'affichage 91

NormVideo 42
NOT 19

OR 74
Ord 72
OutTextXY 101

point virgule 18,20
procédure 24
PutPixel 96

Random 29
Randomize 29
Read 83
ReadKey 28,57
record 85
Rectangle 96
RegisterBGIxxx 104
repeat...until... 25,55
Reset 91
RestoreCrtMode 98
ReWrite 81

Seek 83
SetColor 98
SetTextJustify 101
string 81

tableau 63
TextColor 43,57
TRUE 57
Trunc 38,57
type 54

Unités 23,51
UpCase 46,57

Variable locale/globale 26,38
vérification entrée/sortie . . . 84

while 19,55
with 85
word 29
Wordstar 14,126
Write 18,56,81

XOR 74,75

CONSEILS DE LECTURE

Pour aller plus loin en Pascal et Turbo Pascal

Programmer en Pascal
par Daniel-Jean David et Jean-Luc Deschamps (Editions PSI)

Toutes les notions nécessaires à la programmation en Pascal sont ici expliquées en détail, à l'aide d'exemples et de nombreux exercices résolus à l'intention du programmeur déjà initié à un langage. Les déclarations Pascal concernant les types de données sont traitées avec précision et des notions générales sur les structures de données sont exposées. Sont également étudiés le Pascal UCSD et le Turbo Pascal.

Le livre de Turbo Pascal sur PC et compatibles, version 4
par Frédéric Blanc et Pierre Brandeis (Editions PSI)

Cet ouvrage de perfectionnement contient toutes les commandes de Turbo Pascal, expliquées et illustrées. 16 chapitres permettent de maîtriser progressivement ce langage pour arriver, à la fin du livre à un haut niveau de connaissances : faire de l'assembleur à l'intérieur des routines Pascal, connaître le fonctionnement de Heap et de Pile, maîtriser les pointeurs, etc.

Pour maîtriser son PC ou compatibles

MS-DOS pas à pas, versions 2 à 4
par Alain Pinaud (Editions PSI)

Apprendre les commandes du système d'exploitation MS-DOS en les pratiquant, tel est le but de cet ouvrage. Il aborde les différentes commandes par niveau de compréhension. Après la définition du système, sont vus les commandes essentielles, les nouveautés des versions 3.2 à 4 et les logiciels d'accompagnement. Le lecteur doit être familiarisé avec les principaux termes touchant à l'exploitation de l'ordinateur.

Clefs pour MS-DOS, versions 2 à 4
par Yvon Dargery (Editions PSI)

Accessible à tous les utilisateurs d'IBM/PC ou compatibles, ce mémento permet d'avoir sous la main toutes les informations concernant les commandes des différentes versions de MS-DOS, (version 4 incluse) et divers outils de programmation avancée. Il donne également de nombreuses astuces d'utilisation. L'accès aux commandes, utilitaires et questions spécialisées se fait par l'intermédiaire de leur mot clé.

Clefs pour PC et compatibles - nouvelle édition avec DOS 3.3
par Daniel Martin, Guy Herzet et Philippe Jadoul (Editions PSI)

Ce mémento de références pour PC et compatibles est indispensable au programmeur pour accéder rapidement à toutes les informations : configuration du matériel et de la mémoire, instructions et fonctions du Basic, table des points d'entrée, adresses des variables internes, commandes du système d'exploitation, organisation interne des disques, programmation et brochage des circuits spécialisés.

● Achevé d'imprimer par l'imprimerie Tardy Quercy S.A. - Bourges
Dépôt légal : Mai 1988 - N° d'Imprimeur : 14482 - N° d'Édition : 86595-489-1

Votre avis nous intéresse

Pour nous permettre de faire de meilleurs livres, adressez-nous vos critiques sur le présent ouvrage.

— ***Titre de votre livre :*** .....

— ***Ce livre vous donne-t-il toute satisfaction ?***

.....

.....

— ***Y a-t-il un aspect du problème que vous auriez aimé voir abordé ?***

.....

.....

Si vous souhaitez des éclaircissements techniques, écrivez-nous, ou laissez un message à notre service Minitel (3615 OI PSI), nous ne manquerons pas de vous répondre directement.

Avez-vous déjà acquis des livres P.S.I. ?

lesquels ? .....

.....

qu'en pensez-vous ? .....

.....

Nom ..... Prénom ..... Age .....

Adresse .....

Profession .....

Centre d'intérêt .....

Je désire recevoir un catalogue gratuit et être informé régulièrement des nouveaux livres publiés.

☐ oui ☐ non

Veuillez renvoyer cette page, dûment remplie, aux

Editions PSI
6-10, boulevard Jourdan
75014 PARIS